Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE CARRERA

Título del TFC: Introducción a la Háptica. Nuevos dispositivos de entrada y salida

Titulación: Ingeniería Técnica de Telecomunicaciones, especialidad Telemática

Autores: Vanessa Andreu Toribio
Antonio Torronteras López

Director: Francisco Javier Mora

Fecha: 16 de septiembre del 2015

Título: Introducción a la Háptica. Nuevos dispositivos de entrada y salida

Autores: Vanessa Andreu Toribio
Antonio Torronteras López

Director: Francisco Javier Mora

Data: 01 de septiembre del 2015

Resumen

En el presente trabajo se realiza una revisión del estado del arte de realidad virtual y realidad aumentada, dispositivos de entrada y salida, y tecnología háptica con el objetivo de ver de qué manera contribuye la háptica en la Interacción Persona-Computadora.

El punto de partida del proyecto es el auge que están viviendo tecnologías como realidad virtual o realidad aumentada por un lado y la tecnología háptica por otro. Este auge muestra el gran salto tenológico, en términos de madurez, entre los dispositives audivisuales (HMD, Head-mounted display, cascos o gafes de realidad virtual) y los dispositives hápticos, aún de poco desarrollo para el usuario masivo.

En la actualidad la tecnología parece haber alcanzado la madurez suficiente, en términos de *hardware* y en *software* para la generación de entornos virtuales para el uso práctico por el público masivo y no especializado. Aun así, la sensación de inmersión que obtenemos no siempre resulta del todo completa ya que podemos ver, podemos oír, pero no podemos tocar. La necesidad actual de que estos sistemas sean cada vez más realistas obliga a introducir el sentido del tacto aumentando de forma considerable nuestra inmersión en entornos virtuales.

Así pues, la tecnología háptica, se presenta como un paso natural a todas estas novedades audiovisuales, jugando un papel esencial a la hora de construir y manipular escenarios y objetos virtuales.

Una vez realizada una revisión del estado del arte, se realiza un diseño conceptual de un sistema de entrada y salida basado en háptica para llevar a cabo una implementación práctica de la solución y analizar los resultados obtenidos mediante la realización de ensayos y pruebas.

Title: Haptic Introduction. The new input and output interface

Author: Vanessa Andreu Toribio
Antonio Torronteras López

Director: Francisco Javier Mora

Date: September 01st 2015

Overview

In this project, a review of the state of the art on virtual reality and augmented reality, input and output devices and haptic technology takes place in order to evaluate the role of haptic technology in the Human Computer Interaction (HCI).

The starting point of the project is the broad increasing interest in technologies such as virtual reality or augmented reality, on one hand, and the haptic technology, on the other. We are seeing a great technological leap, mainly through audio visual dispositives (HMD Head- mounted display, headphones or glasses virtual reality) but with some forthcoming haptic devices, which are still unusual for the consumer.

Nowadays, the technology seems to have reached enough maturity, in terms of hardware and software, for generating virtual environments of practical use by massive and unskilled user. Nevertheless, the desire sense of immersion is not yet complete: we can see, we can hear, but we can't touch. The requirements for these systems are becoming more and more demanding on realistic sensations to faithful recreate the immersion, by introducing the sense of touch as essential ingredient in the virtual environments.

Then, the haptic technology comes as a natural step to all these audio visual novelties, as an essential role to build and manipulate virtual objects and scenes.

Once the review of the state of the art is done, a conceptual design of a system based on input and output haptic is made to carry out a practical implementation of the solution and analyse the results obtained through some user tests.

A nuestros padres y hermanos
por su esfuerzo.

A Dani, Vimo y Jordi
por su apoyo.

Y a Carol y Biel
por su paciencia.

ÍNDICE

INTRODUCCIÓN

Este trabajo de fin de carrera se ha realizado en el *Centre Internacional de Mètodes Numèrics en Enginyeria* (**CIMNE**) en colaboración con la *Escola d'*Enginyeria *de Telecomunicació i Aeroespacial de Castelldefels* (**EETAC**) con el objetivo de estudiar y explorar de qué manera contribuye la tecnología háptica sobre la Interacción Persona-Computadora (IPO [1], o HCI, Human Computer Interaction [2]).

Las aplicaciones de Realidad Virtual y Realidad Aumentada son cada vez más frecuentes en áreas como el entretenimiento, medicina o educación ya que pueden utilizarse como herramientas de trabajo o de aprendizaje. Por ejemplo, comienza a ser habitual que en tratamientos de fobias se utilice un entorno virtual en el que un paciente con miedo a conducir puede estar dentro de un vehículo virtual sin necesidad de exponerse a un riesgo innecesario. Sin embargo, este sistema tiene la limitación de transmitir la información solo a través de dos canales (visual y auditivo, a través de gafas o cascos de realidad virtual —Head-mounted Display, HMD [3]—). El paciente de nuestro ejemplo no puede sentir cómo el volante virtual opone resistencia, ni apreciar sus texturas, en definitiva percibir que está conduciendo un vehículo de verdad. Este hecho hace que los sistemas de VR y AR no siempre sean adecuados para simular tareas en las que es necesario que el usuario interactúe físicamente con el entorno. Estas limitaciones parece que podrían superarse en gran medida incorporando dispositivos hápticos, es decir, añadiendo el sentido del tacto a los entornos virtuales.

El presente trabajo prentende contribuir a valorar el papel de la háptica en entornos virtuales mediante dos líneas de actuación: por un lado, a través de una revisión del estado del arte de realidad virtual y realidad aumentada; dispositivos de entrada y salida y tecnología háptica; por otro, realizando la implementación de dos casos de uso de un dispositivo háptico, que ayuden a entender el detalle de los componentes del sistema y su efecto en el usuario final.

El trabajo se estructura de la siguiente manera:

Capítulo 1. Descripción de la necesidad inicial, motivaciones e intereses que llevan al desarrollo del TFC.

Capítulo 2. Se realiza una revisión del estado del arte a modo de marco tecnológico donde nos encontramos en la actualidad referente a la realidad virtual, realidad aumentada y HCI.

Capítulo 3. Se realiza un breve repaso a la evolución de los principales dispositivos de entrada y salida y estado del arte de los mismos.

Capítulo 4. Se introducen y definen los conceptos básicos relacionados con háptica en los seres humanos, su evolución tecnológica, y estado del arte mediante su clasificación y enumeración de los principales dispositivos.

Capítulo 5. Se realiza una breve explicación del diseño de un sistema háptico y selección de un sistema de entrada y salida háptico simple.

Capítulo 6. Se implementa el diseño del sistema de entrada y salida háptico seleccionado en el capítulo anterior y se realiza un pequeño experimento con varios participantes.

Capítulo 7. Se incluyen las principales conclusiones del trabajo realizado destacando el aprendizaje, los progresos y los resultados obtenidos.

Los anexos incluyen información complementaria que no es necesaria para el seguimiento de la descripción del trabajo realizado, pero sí de mucha utilidad para quien desee experimentar o replicar los desarrollos. También incluyen manuales, breves guías de instalación y uso de las aplicaciones. Además del cuestionario utilizado en la realización del experimento.

Anexo I. Cuestionario.
Anexo II. Conexión de WiiMote con el equipo.
Anexo III. Manual de usuario del teclado virtual.
Anexo IV. Manual de usuario del simulador de conducción.
Anexo V. Datasheet Geomagic.
Anexo VI. Datasheet CyberGlove.
Anexo VII. Código fuente del teclado virtual.
Anexo VIII. Código fuente del simulador de conducción.
Anexo IX. Código de la librería de WiimoteLib.dll.

CAPÍTULO 1. MARCO DEL PROYECTO Y MOTIVACION

1.1. Necesidad Inicial

Actualmente estamos asistiendo a un auge de la Realidad Virtual y la Realidad Aumentada, prueba de ello es el interés mostrado por grandes compañías como Sony, Microsoft, Google [4] y Facebook, trabajando e invirtiendo en proyectos como Morpheus [5], HoloLens [6], Magic Leap [7] u Oculus Rift [8] respectivamente. Suele ser habitual que este tipo de empresas realicen inversiones en diferentes áreas pero en este caso hay un factor común, la realidad virtual y realidad aumentada. ¿Es posible que nos encontremos frente a un cambio tecnológico importante?

Como se verá en el capítulo siguiente, hace años que existen dispositivos de Realidad Virtual y Aumentada. Entonces, ¿Por qué se está viviendo este auge ahora? La respuesta a esta cuestión posiblemente se pueda responder si tenemos en cuenta varios factores.

Hardware: Las especificaciones técnicas han ido evolucionando mucho desde los primeros dispositivos de VR y AR. Hay dos mercados muy importantes que han ayudado a alcanzar esta madurez tecnològica:

- El mercado de los videojuegos, con avances en los processadores pero, especialmente destacable, en las tarjetas gráficas (GPU).

- El mercado de la telefonia móvil, que ha ayudado a la miniaturización de sensores, al uso de processadores multi-núcleo muy eficientes energéticamente, etc.

Software: podemos destacar desde la mejora en la compresión de datos, fundamentalmente imágenes hasta el tratamiento de modelos geométricos o el renderizado.

La tecnologia parece haber alcanzado la madurez suficiente, en términos *hardware* y *software* para la generación de entornos virtuales para uso practico por el público masivo y no especializado. Aun así, la inmersión que genera no siempre resulta del todo completa ya que podemos ver, podemos oír, pero no podemos tocar. La necesidad actual de que estos sistemas sean cada vez más realistas obliga a introducir el sentido del tacto aumentando de forma considerable nuestra inmersión en este tipo de entornos.

Así pues, la tecnología háptica, se presenta como un paso natural a todas estas novedades audiovisuales, jugando un papel esencial a la hora de construir y manipular escenarios y objetos virtuales.

Un dispositivo de entrada-salida capaz de entregarnos en tiempo real un *feedback* háptico y sensación de inmersión precisos será la clave para el futuro éxito de la Realidad Virtual.

Esto podria significar un punto de inflexión más allà de los actuales avances en HMD.

1.2. Intereses

1.2.1. Interés CIMNE

Las siglas CIMNE corresponden a Centre Internacional de Mètodes Numèrics en Enginyeria. CIMNE es un centro de investigación público, consorcio entre la Universidad Politécnica de Catalunya y la Generalitat de Catalunya. Tiene como objetivo principal el desarrollo, la divulgación y la transferencia al sector productivo de métodos numéricos novedosos e innovadores.

CIMNE, como instalación generadora y contenedora de conocimientos, se basa en la generación de conocimiento original a través de la investigación, es por ello, que todas las pruebas, todas las mediciones, todo tipo de recopilación de datos realizado en el TFC, sirven para ampliar la base de conocimiento de CIMNE.

Las disciplinas que forman parte del núcleo de la actividad de CIMNE, las simulaciones por odenador y los métodos numéricos en sentido amplio, requieren de avances en computación de altas prestaciones (HPC), últimas tecnologias de visualización, procesado de grandes cantidades de datos, interfícies de usuario amigables y precisas, etc... Componentes todos ellos que también forman parte de los sistemas modernos de realidad virtual y aumentada.

1.2.2. Interés EETAC

La EETAC imparte las ingenierías de telecomunicaciones y aeronáutica. Nuestro proyecto está relacionado directamente con la tecnología informática y electrónica conteniendo una parte de *software* y otra de *hardware*.

1.2.3. Interés Personal

A nivel personal, este trabajo nos ha interesado mucho, dado que hemos utilizado video consolas desde 1985 y hemos visto evolucionar el sector del entretenimiento digital desde sus inicios, desde la Atari hasta las nuevas consolas de Sony o Microsoft. Para nosotros, involucrarnos en un proyecto como éste, significaría formar parte de esta historia, y por qué no, redirigir nuestro rumbo profesional y poder dedicarnos a aquello que realmente nos sentimos más motivados, pudiendo abarcar más conocimientos de dicho sector, poder realizar estudios de investigación, poder crear algo tangible y que pueda tener salida en el mercado.

1.3. Objetivo

El objetivo principal de este TFC es explorar de forma teórica y experimental las capacidades de los dispositivos de entrada y salida hápticos para interaccionar con objetos virtuales.

A continuación se resumen las cuatro fases que se han seguido para conseguirlo:

Fase 1: Repaso del estado del arte de dispositivos hápticos
Fase 2: Diseño conceptual del sistema de entrada y salida basado en háptica
Fase 3: Implementación de la solución
Fase 4: Ensayos y pruebas.

CAPÍTULO 2. REALIDAD VIRTUAL Y AUMENTADA

2.1. Introducción

En 1987, J. Lanier [9], acuñó el término de Realidad Virtual y en 1995, C. Manttea y R. Blade definen la Realidad Virtual como "*Un mundo artificial creado por ordenador, en el cual, el usuario siente que está dentro de él y tiene la capacidad de explorar y manipular objetos de ese mundo"* [10].

En 1992, Tom Caudell, fue contratado por Boeing para facilitar la interpretación de los tableros de configuración del cableado que utilizaban los trabajadores. Para ello, diseñó unas lentes que añadía a los tableros reales unos virtuales generados por ordenador aportando nueva información, fue entonces cuando se le ocurrió decir que "*Los tableros estaban aumentados".* Caudell, fue la primera persona en acuñar el término de Realidad Aumentada [11].

El siguiente capítulo muestra la evolución histórica de estos dispositivos añadiendo, año tras año, unas mejores especificaciones técnicas para permitir crear un mundo virtual de mejor calidad.

2.2. Evolución histórica

La historia de la realidad VR y AR podría remontarse a 1960 cuando Philco Corp. desarrolla, Headsight, el primer HMD (Head Mounted Display), dispositivo que se colocaba en la cabeza y mostraba imágenes gracias a su pantalla, que además disponía de un sistema que permitía hacer seguimiento de la cabeza. Este dispositivo fue utilizado en operaciones de entrenamiento militar.

A finales de los 60, I. Sutherland, crea el primer sistema de AR, al que llamó The Ultimate Display [12]. Se trataba de una pantalla montada sobre la cabeza que permitía al usuario ver un paisaje real con imágenes gráficas superpuestas y conectado a un ordenador. Un ordenador, que por los años 60, procesaba un ciclo de instrucción cada dos mili segundos y disponía una memoria de 4Kbytes.

Fig. 2.1 The ultimate Display [12]

En 1989, E. Howlett se convirtió en la primera persona en comercializar un HMD, CyberFace [13].

En 1992 apareció CAVE (Computer Automatic Virtual Environment) [14], desarrollado por la universidad de Illinois. Se trata de un habitáculo en forma de cubo, donde se proyectan imágenes en las paredes, suelo y techo. CAVE, estaba formado por una sala oscura, necesaria para no obtener luz del exterior y dentro de ésta, otra sala más pequeña donde el usuario se situaba en el interior y podía ver las proyecciones gráficas se realizaban desde la parte posterior de la pared y se mostraban por el otro lado. El usuario, debía tener unas gafas especiales para poder ver en tres dimensiones y así poder ver los objetos flotando en el aire y desde todas sus perspectivas.

Fig. 2.2 Imagen de un usario utilizando CAVE

En 1995, Nintendo lanzó Virtual Boy [15], un dispositivo HMD portátil, dado que no hacía falta conectarlo a ningún ordenador, que producía un efecto 3D muy logrado al haber un desfase entre las dos pantallas que disponía. Las dos pantallas, eso sí, eran monocromáticas. Esta consola, que no tuvo mucho éxito y solo se comercializó en Estados Unidos y Japón.

Fig. 2.3 Virtual Boy de Nintendo [15]

En 1999, los investigadores de las fuerzas Navales empiezan a trabajar en campos de guerra con sistema de realidad aumentada, retomando el modelo original de 1992 utilizado por los soldados [16].

Entre 1999 y 2004 apenas hay bibliografía. Como podemos ver en el gráfico 2.5, no hubo muchos desarrollos conocidos.

En 2004 un grupo de investigadores alemanes llevan la tecnología de la realidad aumentada a los teléfonos móviles, pero solo para usuarios especializados. Presentaron un sistema para el seguimiento de marcadores 3D a través de la videcámara del móvil utilzando AR [17].

En 2008 apareció al público Wikitude [18], la primera aplicación de AR, para el usuario no especializado, que podía utilizarse a través de un smartphone. Esta aplicación permitía al usuario poder ver a través de su pantalla capturando imágenes reales mediante la cámara incorporada, añadiendo capas de información obtenidas a través de su geoposicionamiento. Estas capas de información variaban en función de la captura que se realizara, pero podía facilitar lugares cercanos al usuario o el nombre y la descripción de monumentos u obras de arte.

En Abril de 2012, se empieza a probar un proyecto desarrollado por Google, este proyecto se conoce con el nombre de Google Glass y son unas gafas del tipo AR. La comercialización se produjo en Abril del 2013. Estas gafas crearon mucha expectación y parece ser el experimento más notable para masificar el uso de gafas de AR (más allà del laboratorio), con controversias sobre temas de privacidad [19]. Otro factor que fue de gran aceptación por el usuario, es el control total de dispositivo mediante comandos de voz. Con tan solo decir "*Ok, Glass"* se activaban y el usuario solo debía enviar instrucciones tales como: "*Dirígeme hasta la calle... ", "Envía un email a...”*. Podíamos añadir información adicional relacionada con lo que estábamos viendo, realizar vídeo llamadas a través del cliente de mensajería predeterminado, Hangouts. Muchos se preguntarían cuál es la diferencia con respecto a utilizar un Smartphone. La respuesta es sencilla: es un dispositivo con el que podíamos interactuar con la voz y a su vez, podíamos ver perfectamente lo que ocurre a nuestro alrededor, sin necesidad de mirar la pantalla de nuestro smartphone. Tal es el impacto tecnológico, que se han realizado intervenciones médicas retransmitidas a través de las gafas pudiendo intervenir a distancia. Además como en los

teléfonos inteligentes actuales, disponemos de giroscopio, acelerómetro, sensores geomagnéticos y sensores de luz.

Este dispositivo no llegó a comercializarse en Europa. El motivo no parece ser muy conocido, pero lo que sí se ha publicitado es la fuerte apuesta de Google por otro nuevo dispositivo de AR, las Magic Leap ya mencionadas anteriormente.

Fig. 2.4 Google Glass (**Fuente:** Google)

Como se observa en la figura 2.5, entre 1990 y 2000 hubo un aumento de desarrollos y proyectos relacionados con VR y AR. No obstante, no es hasta 2013 donde realmente se puede comprobar el repunte de estas tecnologías [20].

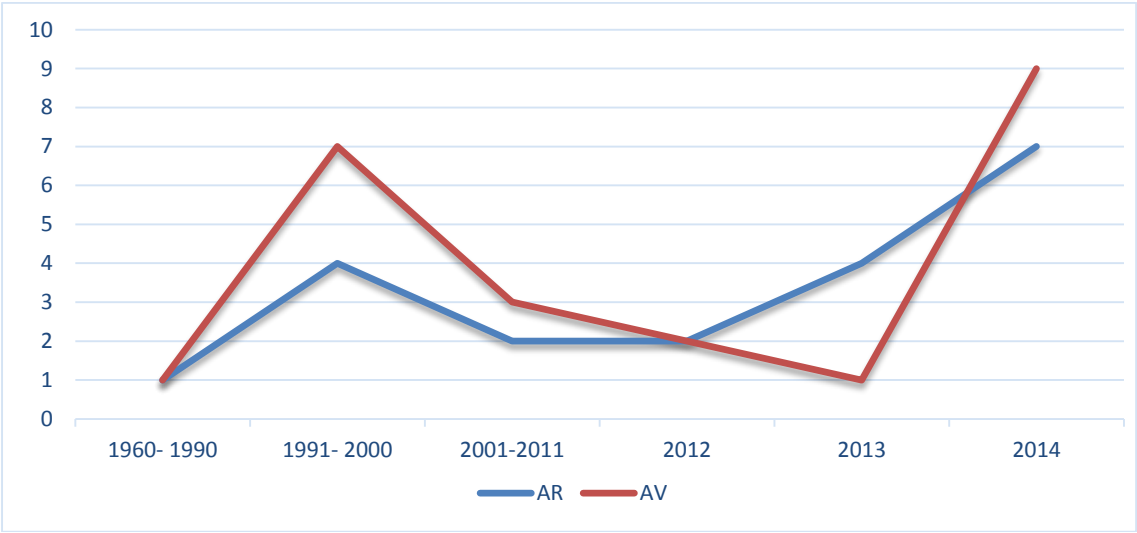


Fig. 2.5 Evolución temporal de desarrollos de VR y AR (**Fuente:** Elaboración propia)

2.3. Estado del Arte

Como hemos visto revisando la evolución histórica, cada vez más, los dispositivos disponen de características computacionales más potentes (procesadores, memoria, etc), lo que permite recrear mundos virtuales de mayor calidad. Los siguientes dispositivos que destacamos a continuación son los que actualmente encabezan la carrera tecnológica en HMD.

En Julio de 2012, P. Luckey presenta Oculus Rift [8], un proyecto que se dio a conocer a través de Kickstarter [21]. Este periférico pretende mejorar los dispositivos HMD para la realidad virtual que existen en la actualidad. Oculus fue adquirido por Facebook en 2014 [22], y ahora son ellos quien están financiando el desarrollo, para poder ser lanzando durante el primer trimestre de 2016. Oculus Rift es la evolución de los primeros dispositivos de VR, el principio básico y finalidad sigue siendo el mismo, pero a diferencia de estos, muestra el salto tecnológico que comentamos en la introducción, proporcionando al usuario la sensación de mayor inmersión dentro del juego.

Fig. 2.6 Oculus Rift (**Fuente:** Facebook)

En Marzo del 2014, Sony Presenta Project Morpheus, un HMD para realidad virtual que permite visualizar en su pantalla AMOLED de 5.7 pulgadas de alta resolución (1080p) un entorno virtual generado gracias a la potencia de su video consola de última generación PS4 que presenta un procesador de AMD de 8 núcleos, 8 GByte de memoria RAM compartida por la tarjeta gráfica AMD, específica de la consola de SONY. Morpheus lleva incorporado un sensor de seguimiento de posición de la cabeza, como parece siendo habitual en estos dispositivos desde los años 60. Este dispositivo saldrá al mercado durante el 2016, al igual que Oculus.

Fig. 2.7 Morpheus VR (**Fuente:** Sony)

En enero del 2015, Microsoft, uniéndose a la tendencia del resto de las grandes compañías, presenta HoloLens, su propio HMD para realidad aumentada. A que a diferencia del resto, no genera imágenes en 3D, sino que lo que muestra son hologramas y, al igual que las Google Glass, no necesita estar conectado a ningún ordenador, videoconsola o Smartphone, dado que dispone de una tarjeta gráfica y procesador incorporados en las lentes. Aún no se han publicado sus especificaciones técnicas pero si algunas de sus funcionalidades, por ejemplo, realizar videoconferencias a través de skype, con funciones extra (interacción de los participantes con esquemas u otros componentes que comparten como si fueran flotantes, superpuestos a la realidad.

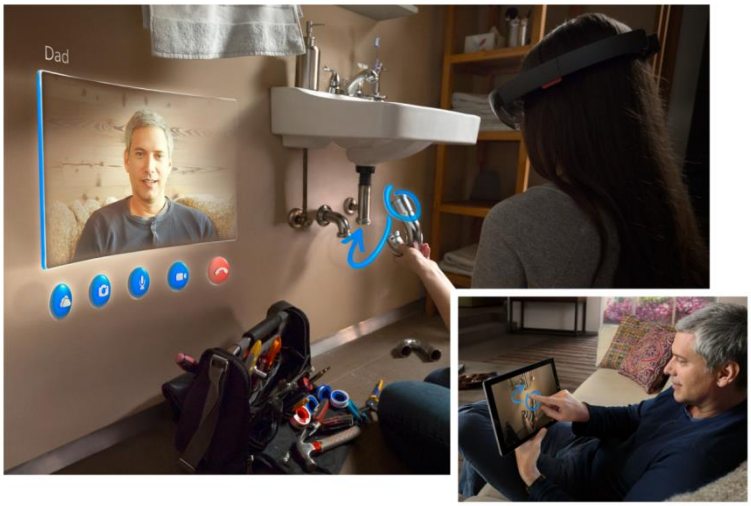


Fig. 2.8 Interacción HoloLens (**Fuente:** Microsoft)

En enero del 2015 se da a conocer Magic Leap [7], un dispositivo de AR que pretende revolucionar el mundo del entretenimiento. Al igual que en el caso anterior, todavía no se conocen sus características técnicas, ni tampoco la fecha de lanzamiento, no obstante, ha sido destacado el interés de empresas como Google y Qualcomm, y de su inversión por más de quinientos millones de dólares, lo cual parece anunciarnos grandes desarrollos tecnológicos en esta área.

Fig 2.9 Magic Leap en acción. (**Fuente**: CNN [23])

Hasta ahora, los dispositivos de AR eran básicamente capas de información superpuestas a lo que vemos en realidad. Este dispositivo, aparte de permitir el acceso al correo, participar en videoconferencias, ver videos de Youtube, también parece ofrecer la capacidad de interaccionar con objetos virtuales como si fueran reales, fundiendo las modalidades de juego real y virtual en en un mismo escenario. ¿Estamos ante una convergencia entre los entornos virtuales y aumentados?

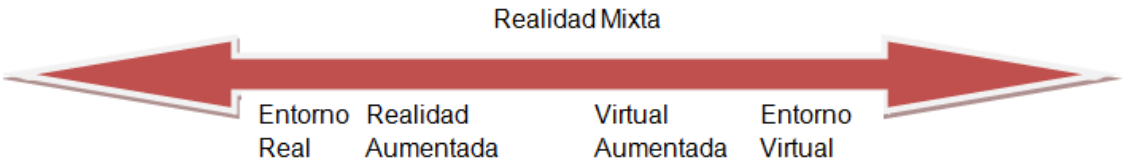


Fig. 2.10 Convergencia VR/AR (**Fuente:** Elaboración propia)

En Mayo del 2015, se da a conocer FOVE [24] en la plataforma kickstarter, ¿En qué se diferencia al resto? ¿Qué le hace especial? La respuesta es sencilla, sigue siendo un sistema de VR, al más puro estilo Oculus Rift, pero la diferencia radica en que es posible utilizarlo directamente como dispositivo de entrada a través de los ojos del usuario. Este dispositivo dispone de un sensor que es capaz de capturar el movimiento del ojo y así poder utilizarlo como sustituto al ratón.

Durante el último año, noticias como la adquisición de Oculus Rift por parte de Facebook o la adquisición de Metaio por parte de Apple [25] parecen ser un claro indicio del interès estratégico de la realidad virtual y augmentada para las grandes del sector tecnológico. En la figura 2.11 podemos ver el estudio realizado por Kzero [26], donde la previsión de consumo de *hardware* y *software* aplicado a la realidad virtual y aumentada crecerá al 2014. De confirmarse esta tendencia, parece raonable esperar, también, que vayamos a presenciar grandes saltos tecnológicos en esta àrea

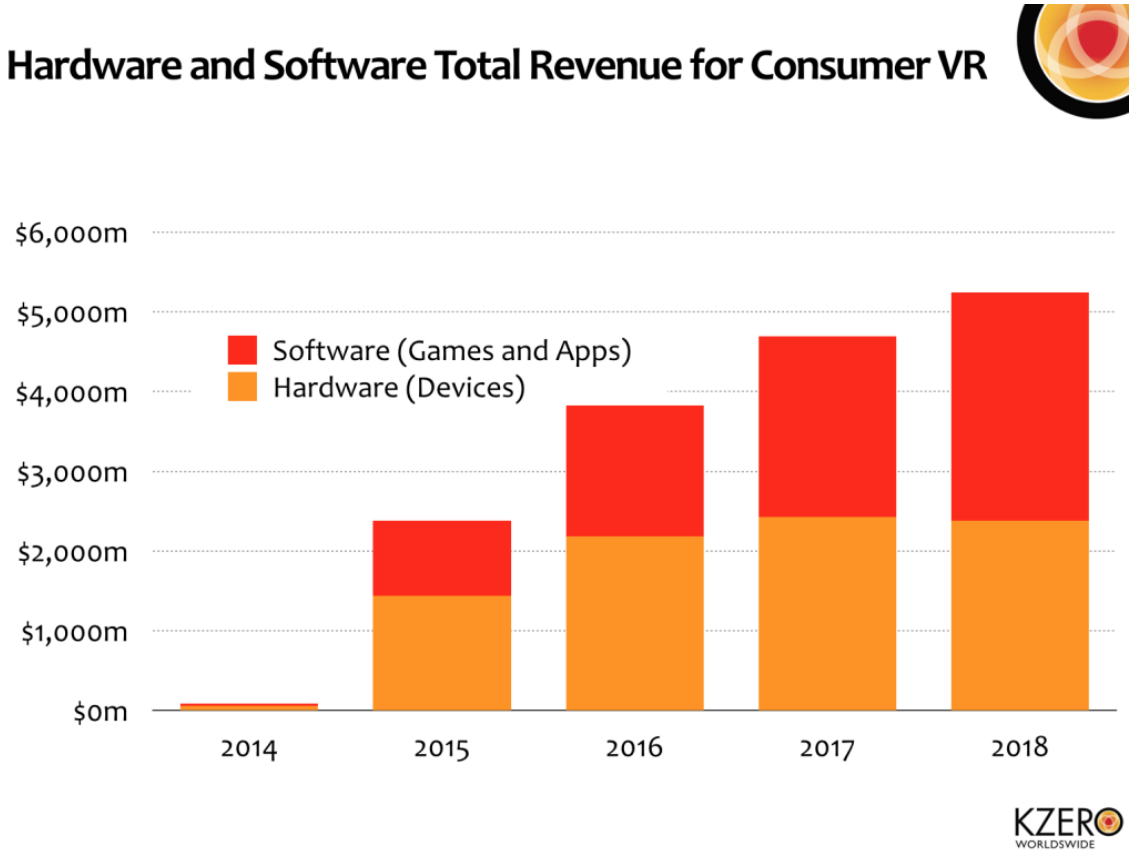


Fig. 2.11 Estimación de ingresos de *Hardware* y *Software* para la venta de VR. (**Fuente**: Kzero Worldswide)

Como indica la figura 2.11, en los próximos años, habrá un aumento tanto de dispositivos de realidad VR/AR como de aplicaciones para los mismos, lo cual indica que las grandes compañías se están involucrando cada vez más en esta tecnología y que la VR será una tecnología rutinaria en el día a día de los consumidores no especializados de contenido digital (Consumo masivo).

Con estas previsiones de ingresos provenientes del consumidor, es razonable pensar que las empresas invertirán más en innovación y avance de la tecnología. De ser cierto, ¿qué otros grandes avances podremos esperar?, ¿tal vez será posible utilizar unas lentillas para poder visualizar un entorno virtual o aumentado a través del ojo del ser humano?

CAPÍTULO 3. DISPOSITIVOS DE E/S

3.1. Introducción

Podríamos decir que la Interacción Persona-Computadora (IPO, HCI *Human-Computer Interaction*), en términos generales, es la disciplina que estudia el intercambio de información entre las personas y las computadoras. Esta se encarga del diseño, evaluación e implementación de los aparatos tecnológicos interactivos, estudiando el mayor número de casos que les pueda llegar a afectar. El objetivo es que el intercambio sea más eficiente: minimizar errores, incrementar la satisfacción, disminuir la frustración y, en definitiva, hacer más productivas las tareas que rodean a las personas y los computadores [1].

En el presente capítulo, entenderemos los periféricos o dispositivos de entrada y salida como el **medio** por el cual máquina y usuario pueden comunicarse. Existen tres grupos de este tipo de dispositivos, los de entrada, los de salida y los de entrada y salida.

Un periférico de entrada es aquel elemento que permite al usuario, mediante un medio externo, introducir datos en la máquina.

Un periférico de salida es aquel canal que permite a la máquina mostrar la información al usuario mediante el medio externo.

Un periférico de entrada y salida es el medio que permite introducir datos en la máquina y además enviar información al usuario por el mismo canal.

3.2. Evolución histórica

Los dispositivos de entrada y salida fueron creados para facilitar la comunicación entre máquina y usuario y poder recibir y enviar información. A lo largo de estos años, estos dispositivos han sufrido una clara evolución manteniendo siempre la misma finalidad.

Fig. 3.1 Principales dispositivos de entrada y salida [27] (**Fuente:** softhardaa)

A continuación listamos los principales dispositivos de entrada:

- Tarjetas perforadas: El primer periférico de entrada que permitía al usuario introducir programas en el ordenador. Originalmente sólo se codificaba información numérica.
- El teclado: Fue uno de los primeros dispositivos de entrada. Permiten al usuario introducir datos alfanuméricos en el ordenador.
- El ratón: Es un periférico de entrada, que se empezó a utilizar para facilitar el manejo del entorno gráfico (datos gestuales, coordenadas relativas, básicamente el movimiento del cursor y los botones de activación de determinados comandos).
- *Joypad*/*Joystick*: Son periféricos de entrada que el usuario utiliza para poder manejar simuladores o videojuegos. Envía datos con información sobre coordenadas relativas o botones de activación de comandos para emular a mandos de vehículos para navegación aérea o terrestre.
- Micrófonos: Permite al usuario una entrada de datos a través de la voz. El dato que se envía es el audio.
- Tabletas digitalizadoras. Son periféricos que permiten al usuario introducir dibujos o gráficos realizados a mano con la ayuda de una *stylus* ("lápiz digital"), tal y como se haría con lápiz y papel. Envía datos al sistema con información acerca de las coordenadas absolutas.
- *Touchpad*: Es un sistema cuya función es la misma que la del ratón. Envía datos con información acerca de coordadas relativas, botones de activación y datos gestuales.
- Escáner: Permite la entrada de información a través de la digitalización de un documento (imagen).

- *WebCam*: Capta las imágenes y sonido del exterior para enviarlos al sistema.

Y los principales dispositivos de salida:

- Led's: El primer dispositivo de salida eran unas luces que se encendían y apagaban cuando se realizaban ciertas acciones, como por ejemplo acceder a determinadas posiciones de memoria.
- Impresoras: El segundo dispositivo de salida se creó en los años 60, y se utilizaba para mostrar al usuario la información de cualquier programa (típicamente los resultados de determinados cálculos) y solo podía imprimir texto [28].
- Monitor. En los años 70 aparecieron los monitores, Medio por el cual el ordenador podía comunicarse con el medio externo de una forma más económica. El ordenador muestra información al usuario de todo tipo, ya sea a través de líneas de texto, gráficos, etc. Los primeros monitores eran monocromáticos y presentaba la información con caracteres alfanuméricos.
- Altavoces. Medio por el cual muestra información al usuario en forma de audio.

También podemos encontrar periféricos que funcionan como dispositivos de entrada y salida; son aquellos que permiten una comunicación bidireccional entre el ordenador y el medio externo. El mejor ejemplo de este tipo de periférico es el MODEM, que permite al ordenador enviar y recibir información del exterior sin interactuar directamente con el usuario.

3.3. Estado del arte

A partir de la clasificación preliminar del anterior apartado es importante notar que, con el paso de los años, un dispositivo que era exclusivamente de salida, se ha convertido también en un dispositivo de entrada. Un buen ejemplo de ello son las pantallas de *smartphones* y *tablets* que son dispositivos con los que además de visualizar la información, podemos emplearlos para introducir datos (gestos o escritura), además de recibir una pequeña vibración ante determinadas acciones (al pulsar alguna de las teclas virtuales, por ejemplo), lo que constituye uno de los mas claros ejemplo de dispositivos hápticos actuales.

Otro claro ejemplo de dispositivo de entrada y salida que ha evolucionado considerablemente durante las últimas décadas son los mandos controladores. Este tipo de dispositivo, permite al usuario enviar una instrucción a la máquina, actuando como dispositivo de entrada y en función del resultado, recibir una respuesta en forma de vibración, actuando así como dispositivo de salida.

Otros dispositivos de entrada y salida de datos relacionados con la realidad virtual, son los guantes hápticos que permiten la comunicación bidireccional entre el ordenador y el usuario. El guante envía instrucciones de posición al ordenador, para poder recopilar información del movimiento de las manos y de los dedos del usuario y así poder integrarlo en un escenario virtual para después ofrecer una realimentación táctil que nos permita experimentar las colisiones o sensaciones que provienen de la interacción con objetos virtuales.

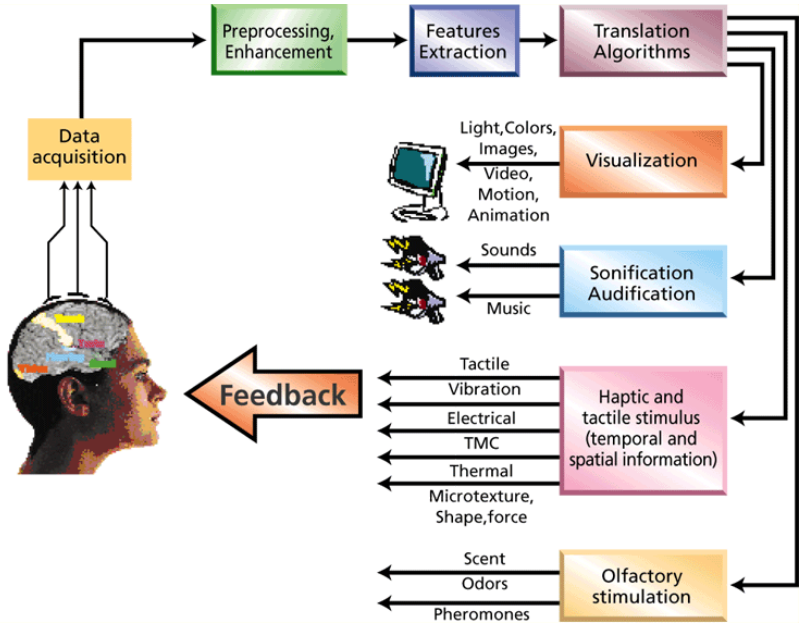


Fig. 3.2 Clasificación de dispositivos de entrada/salida en función del *feedback* que se recibe (**Fuente:** Brain Riken) [29]

También existen proyectos de dispositivos de salida que permiten al usuario sentir olores. Es el caso de FeelReal [30]. Un periférico en forma de máscara que permite al usuario disponer de una serie de olores en función de lo que está viendo, ya sea una fruta o una flor. Este tipo de dispositivo da al usuario una mayor sensación de inmersión en el entorno virtual.

Los HMD (gafas y cascos), asi como otros dispositivos de VR y AR son dispositivos de entrada/salida que no volveremos a describir pues ya se presentaron en el capitulo anterior.

Cada día más, debido a la convergencia tecnológica en computación, robótica, sensores, etc… los dispositivos de entrada y salida crecen en funcionalidades, impensables años atrás, ¿Seremos nosotros directamente los periféricos de entrada y salida? A fecha de hoy ya se está trabajando en ese sentido, empresas como Tekever Technologies for the Evernet [31] con el proyecto Brainflight [32], aseguran que en un futuro cercano cualquier persona será capaz de pilotar drones, aviones o actividades cotidianas como conducir, a través de las ondas cerebrales.

Fig. 3.3 BrainFlight o cómo pilotar un dron con la mente (**Fuente:** airshare.co.nz)

CAPÍTULO 4. DISPOSITIVOS HAPTICOS

4.1. Introducción

Como se ha comentado en el marco introductorio del proyecto (cap. 1), la Realidad Virtual es un campo que está experimentando un mayor crecimiento, producto de las mejoras y avances que se han hecho durante los últimos años. Sin embargo, y a pesar del alto grado de realismo que se obtiene con estas aplicaciones, la VR tiene limitaciones ya que generalmente la información sólo es transmitida a través dos canales: el canal visual y el canal auditivo. Esto permite al individuo sumergirse en el mundo virtual (en el sentido inmersivo) pero, ¿le permite interactuar? En términos generales, los usuarios suelen emplear dispositivos tipo mandos controladores o gestuales, pero no aquellos que les permitiría "sentir" objetos virtuales específicos más allá de un escenario envolvente.

En medicina por ejemplo, los cirujanos pueden ensayar técnicas y realizar operaciones sobre pacientes virtuales. Sin embargo, y a pesar del alto grado de realismo de estas aplicaciones, el hecho de transmitir información a través de la vista y el oído hace que estos sistemas de VR no sean adecuados para simular tareas en las que es necesario que el usuario interactúe físicamente con determinados elementos, cuerpos o piezas.

Como respuesta a esta limitación y con el objetivo de aumentar la sensación de inmersión del usuario, a comienzos de los años 90 empezaron a desarrollarse los primeros dispositivos hápticos [33].

Según el filósofo Bertrand Russell "…es el tacto lo que nos proporciona sentido de realidad y toda nuestra concepción de lo que existe fuera está basada en el sentido del tacto". Así pues, podemos definir el término Interfaces Hápticos como el conjunto de dispositivos que permiten al usuario tocar, sentir o manipular objetos simulados en entornos virtuales y sistemas teleoperados, proporcionándole con ello sensación de inmersión y la posibilidad de interactuar con el medio virtual. En otras palabras, establecer una comunicación bidireccional en tiempo real entre el usuario y el medio virtual. [34]

4.2. Definiciones

Es cada vez más frecuente escuchar hablar de términos como háptica, quinestesia, kinestesia, sin embargo, ninguno de estos términos aparecen en el Diccionario de la Real Academia de la Lengua Española [35], no obstante, sí que podemos encontrar este tipo de terminología en la Wikipedia.

La RAE si reconoce la palabra cinestesia, pero no sus equivalentes quinestesia y kinestesia que también se usan frecuentemente.

cinestesia (Del fr. cinesthésie, y este del gr. κίνησις, movimiento, y αἴσθησις, sensación).
1. f. Psicol. Percepción del equilibrio y de la posición de las partes del cuerpo.

cenestesia (Del gr. κοινός, común, y αἴσθησις, sensación).
1. f. Psicol. Sensación general de la existencia y del estado del propio cuerpo, independiente de los sentidos externos, y resultante de la síntesis de las sensaciones, simultáneas y sin localizar, de los diferentes órganos y singularmente los abdominales y torácicos.

sinestesia (De sin- y el gr. αἴσθησις, sensación).
1. f. Biol. Sensación secundaria o asociada que se produce en una parte del cuerpo a consecuencia de un estímulo aplicado en otra parte de él.
2. f. Psicol. Imagen o sensación subjetiva, propia de un sentido, determinada por otra sensación que afecta a un sentido diferente.

¿Qué relaciona los términos tacto, Kinestesia y háptica? Tradicionalmente se han diferenciado tres modos de procesar la información sobre objetos y patrones aprendida a través del sentido del tacto: Percepción táctil, percepción kinestésica y percepción háptica. [36]

- **Percepción táctil**. Hace referencia a aquella información que se obtiene a través de la piel y que está producida a través de un estímulo.
- **Percepción kinestésica**. Es la información que obtenemos de los músculos y tendones.
- **Percepción háptica**. Cuando percepción táctil y kinestésica se combinan para proporcionar información sobre los objetos reales o virtuales. Es decir, la combinación de la información extraída a través del sistema cutáneo y motor. Esta es la forma habitual de percibir los objetos cuando utilizamos el sentido del tacto de manera activa y voluntaria.

En resumen, podemos definir el sistema háptico como aquel que adquiere información a partir de la percepción táctil y la percepción kinestésica. Es decir, gracias al sentido táctil somos capaces de recibir información a través de los receptores y fibras nerviosas situados en la piel sobre los estímulos de la superficie externa de nuestro cuerpo, y gracias al sentido kinestésico seremos capaces de determinar la posición, estática o dinámica, de las distintas partes del cuerpo mediante los mecanorreceptores que se encuentran en articulaciones, músculos y tendones. Por ello, la realización de prácticamente cualquier tarea de manipulación y exploración de objetos realizada con las manos implica la necesidad de recibir información de ambos sentidos.

4.3. Percepción háptica en los seres humanos

Nos parece interesante incluir un breve capítulo en el que se hable a grandes rasgos de la piel, su fisiología y como se interpretan algunas de las sensaciones que percibimos a través del sentido del tacto y por ende en los sistemas hápticos.

Al igual que con los dispositivos de entrada-salida hápticos, una de las características que diferencia al sistema háptico humano del sistema visual o auditivo es que es bidireccional. Es decir, somos capaces tanto de sentir como ejercer fuerzas. Por lo tanto, antes de diseñar interfaces hápticos eficientes es importante conocer como los seres humanos sienten y ejercen dichos estímulos.

Podemos definir el sentido del tacto como aquel sentido que mediante una serie de mecanismos sensoriales nos permite percibir cualidades tales como aspereza, temperatura, dureza, forma, presión, etc. El sentido del tacto reside principalmente en el órgano de la piel, donde finalizan distintas clases de receptores nerviosos encargados transformar los estímulos del exterior en información susceptible para ser tratada e interpretada por el cerebro. [37]

Se habla de un sexto sentido estrechamente relacionado con el tacto. Como hemos comentado en el punto anterior, se trata del sentido Kinestésico, también llamado Quinestesia o Cinestesia. La Kinestesia se puede definir como la habilidad de sentir las posiciones y movimientos de nuestros músculos y articulaciones, es decir, cómo se percibe la posición y el equilibrio de las diversas partes del cuerpo. Esta habilidad hace posible que seamos capaces de coordinar nuestros movimientos para poder caminar, hablar y usar nuestras manos [38]. La kinestesia nos permite, por ejemplo, que podamos tocarnos la punta de la nariz con los ojos cerrados, pero, ¿sería posible rascar nuestra nariz virtual sin necesidad de mirar?

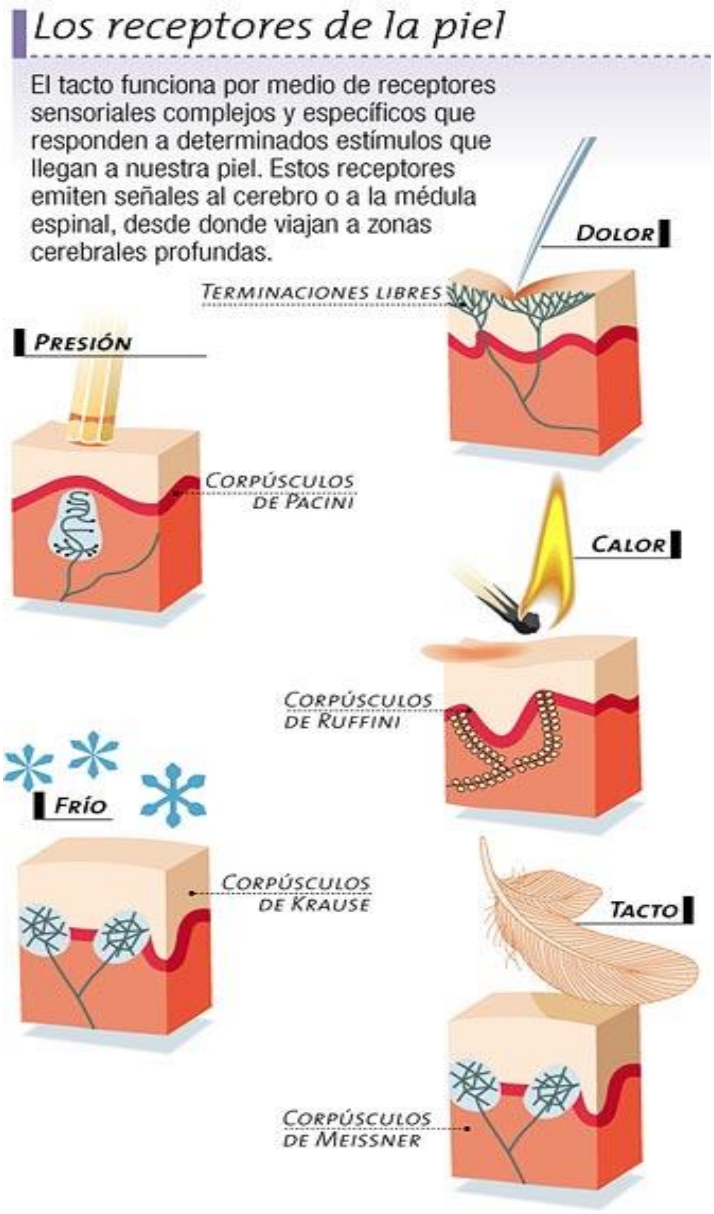


Fig. 4.1 Receptores i fibras nerviosas de la piel. [39] (**Fuente:** M. Garat)

La piel forma parte del sistema somatosensorial que a su vez forma parte del sistema nervioso. A grandes rasgos, la piel está formada por dos capas de receptores táctiles, la superior y encargada de la sensibilidad de precisión y la inferior, menos fina y precisa pero más resistente. A estas dos capas se les debe sumar una red de sensibilidad térmica. A lo largo de toda la piel se distribuyen los receptores táctiles o corpúsculos que podemos dividir en [40]:

- Corpúsculo de Meissner. Son receptores situados en la superficie de la dermis con un campo receptivo pequeño. Su función es la recepción de las sensaciones de tacto ligero, vibraciones y presentan una mayor sensibilidad con excitaciones que se encuentran entre 20 y 50 Hz de frecuencia. Los corpúsculos de Meissner no son receptores aptos para

detectar presiones sobre la piel, sino que son los encargados de detectar cambios en la textura de los objetos.

- Corpúsculos de Krause. Son una variación anatómica de los corpúsculos de Meissner. A diferencia de los anteriores, estos se encuentran situados a más profundidad en la hipodermis y permiten que seamos capaces de registrar sensaciones de frío debido a bajas temperaturas.

- Celdas de Merkel. Como en el caso de los corpúsculos de Meissner, las celdas de Merkel son receptores con un campo receptivo pequeño pero situados más superficialmente, justo en el límite de la epidermis. Su función principal es detectar la presión, patrones y textura de los objetos siendo más sensibles a vibraciones a bajas frecuencias que oscilan entre 5 y 15 Hz.

- Terminaciones de Ruffini. Estas terminaciones se encuentran en las zonas más profundas de la dermis y son las encargadas de detectar cambios en la temperatura relacionados con las sensaciones de calor. A diferencia de las celas de Merkel y los corpuscúlos de Meissner, presentan un campo receptivo grande, es decir, pueden abarcar buena parte de la mano (como un dedo o la palma).

- Corpúsculos de Pacini. Están situados en la dermis, justo entre los corpúsculos de Ruffini y los de Meissner. Estos se encargan de responder a vibraciones intensas y presiones mecánicas. Es por ello, que las vibraciones a diferencia de los casos anteriores son de frecuencias elevadas que oscilan entre 60 y 400 Hz, aunque el rango óptimo de los seres humanos se encuentra entre 200-250 Hz. Sus campos receptivos son grandes y a parte de ser los receptores del sentido vibratorio también se pueden emplear para enviar información sobre el movimiento de las articulaciones.

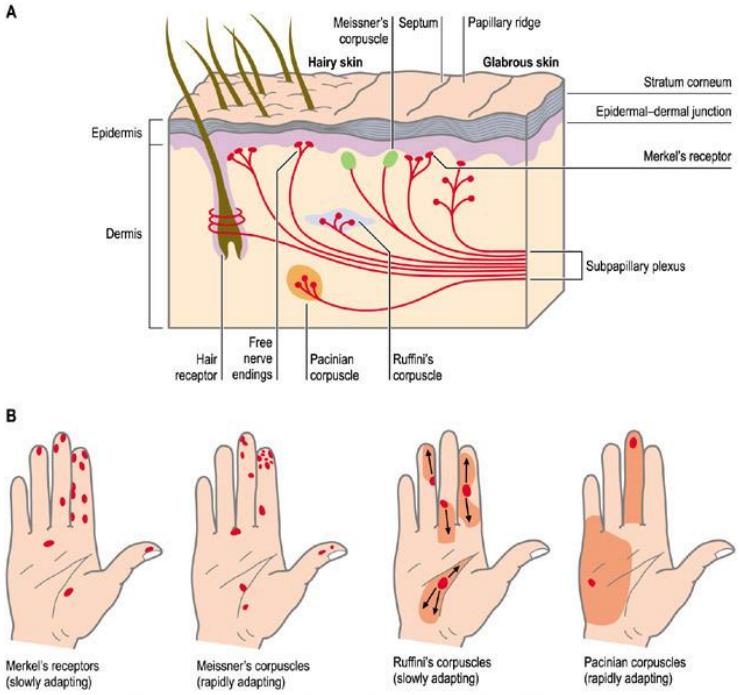


Fig. 4.2 Situación en la mano humana de los principales receptores cutáneos. [41] (**Fuente:** Dr. Luis Fernando Pacheco B.)

A la hora de diseñar dispositivos hápticos, es importante tener en cuenta que los seres humanos somos capaces de sentir los estímulos táctiles y cinestésicos mucho más rápido de lo que somos capaces de responder o actuar.

En la figura 4.3 podemos ver la clasificación frecuencial frente a los estímulos táctiles que Shimoga [42] definió en 1992:

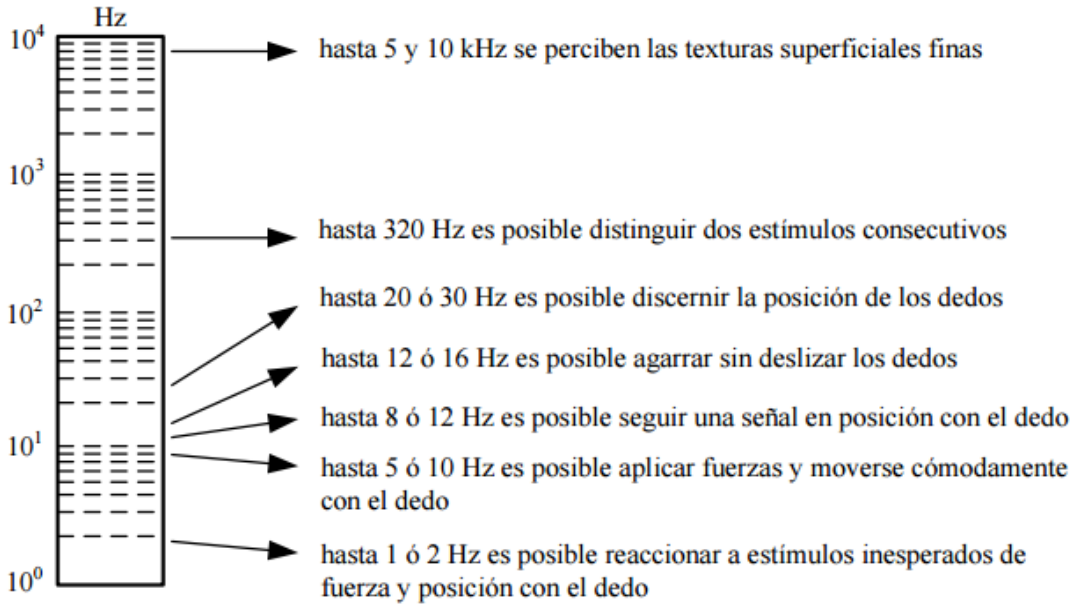


Fig. 4.3 Clasificación frecuencial de los estímulos (**Fuente:** Shimoga)

Teniendo en cuenta estos rangos de frecuencia, la mayoría de diseñadores de dispositivos hápticos, optan por utilizar frecuencias de muestro de 1KHz, dado que les permite simular la mayoría de estímulos táctiles.

4.4. Estado del arte

Como se ha comentado anteriormente, la mayoría de sistemas máquina-usuario tienen un flujo de información unidireccional. Es decir, el usuario puede recibir información a través de los sentidos de la vista y el oído, y puede interactuar con el entorno a través del uso de diferentes periféricos como el ratón, teclado, etc. Esta interacción se muestra de forma unidireccional del usuario hacia el entorno. En cambio con el uso de interfaces hápticas, esta comunicación se convierte en bidireccional. La inclusión del sentido del tacto, hace que la inmersión con el entorno virtual sea más intuitiva y real, proporcionando al usuario información adicional que mejora la sensación de inmersión.

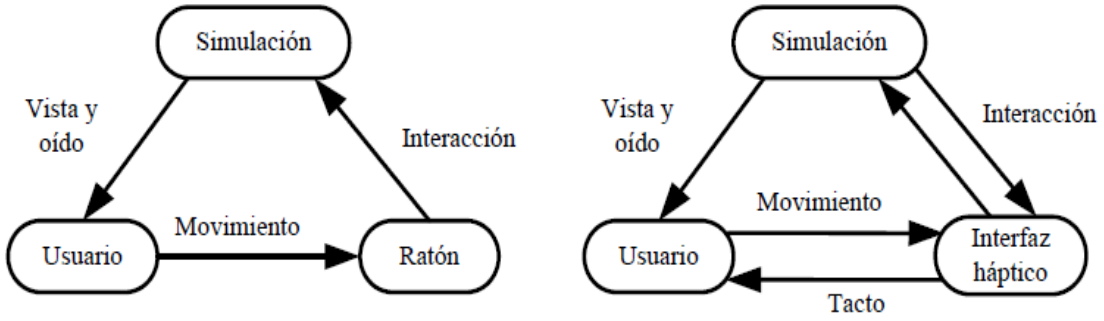


Fig. 4.4. Sistema bidireccional del dispositivo háptico (**Fuente:** Iñaki Diaz)

En la actualidad, y debido al gran auge del mercado de tablets y smartphones, la empresa Microsoft está investigando incluir teclados hápticos en las pantallas táctiles [43]. Aseguran que aunque hoy en día parezca que la vista es lo único que necesitamos para controlar una interfaz (unidireccional), la experiencia puede cambiar considerablemente si se añade una respuesta táctil (bidireccional). Al fin y al cabo, somos seres multisensoriales capaces de recibir estímulos de varios sentidos al mismo tiempo y combinarlos entre sí. Así pues, con la inclusión de los teclados hápticos, Microsoft ha podido demostrar en sus estudios que tanto la rapidez de escritura como la sensación de la misma, son mucho mayores a la de un teclado táctil convencional.

Para profundizar más en los diferentes tipos de dispositivos hápticos que existen o han existido, en los siguientes subapartados realizaremos una clasificación en función de su finalidad y según el tipo de respuesta háptica. Asimismo, analizaremos los dispositivos más significativos del momento.

4.4.1. Clasificación Funcional

En la actualidad los sistemas hápticos se están implantado en diferentes áreas como medicina, entretenimiento, aeronáutica, etc. y se espera que durante los próximos años la necesidad y uso de estos sistemas aumente considerablemente por parte del consumidor [44].

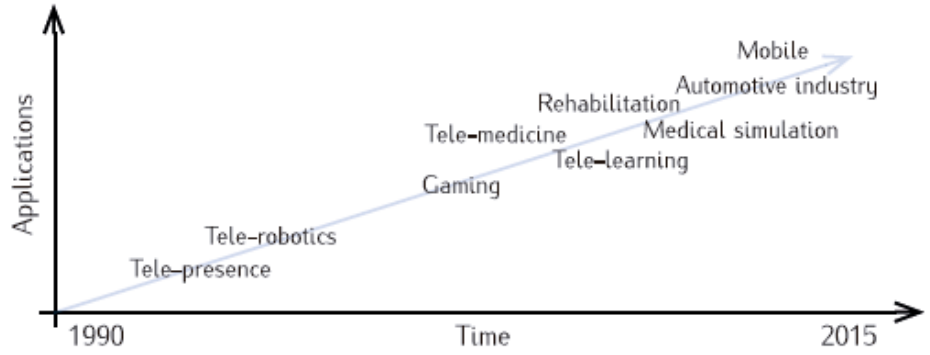


Fig. 4.5 Evolución de los dispositivos hápticos según clasificación funcional (**Fuente:** Pablo Ciáurriz)

4.4.1.1. Entretenimiento

Esta tecnología lleva tiempo aplicándose en videojuegos y simuladores que permiten al usuario sentir y manipular objetos virtuales. Existen multitud de juegos tanto con joysticks o tabletas táctiles en los que el usuario puede ser capaz de sentir las irregularidades a través de vibración. En los últimos tiempos, también esta tecnología se ha aplicado a videoconsolas, añadiendo por ejemplo fuerzas al propio mando que hace que la inmersión del jugador sea mayor en juegos de billar, tenis, etc.

El desarrollo de controladores de juegos con respuesta háptica en forma de vibración hace años que existe. No obstante, empresas como Valve han desarrollado mandos (Steam Controler [45]) donde los actuadores hápticos van situados a ambos lados proporcionando de esa manera vibraciones más precisas y de gran fidelidad en microsegundos.

4.4.1.2. Medicina

La medicina es uno de los campos donde más se ha investigado la aplicación de la tecnología háptica. Hoy en día existen diferentes interfaces hápticos que ayudan en campos de simulación médica y que resultan de gran utilidad a la hora de minimizar los daños que podrían ocasionar diferentes técnicas invasivas.

Poder crear entornos de formación en el campo de la medicina y permitir realizar simulaciones de operaciones quirúrgicas representará un gran avance en el modo de enseñanza y experimentación de la medicina actual, siendo un complemento ideal contribuyendo a la adquisición de destrezas básicas y mejorando el aprendizaje de las complejas técnicas quirúrgicas.

Recientemente, Cambridge Research & Development [46] ha creado un dispositivo (Neo) que permite transferir la retroalimentación a otra parte del cuerpo del cirujano, evitando de esta forma el peligro que presenta la instalación de vibradores o motores en los dispositivos hápticos usados en intervenciones quirúrgicas. De esta forma, se está estudiando la posibilidad de que el cirujano a la vez que utiliza material quirúrgico sobre el paciente, reciba la retroalimentación en la parte posterior de la cabeza. Es decir, conseguir que el dispositivo aplique una presión proporcional a la que siente el material quirúrgico.

4.4.1.3. Teleoperación

Aplicar la háptica para poder controlar herramientas de forma remota y obtener información táctil de que sucede en el otro extremo del robot resulta de gran ayuda, tanto en sistemas quirúrgicos como en sistemas industriales.

A principios del siglo XXI la empresa norteamericana Intuitive Surgical [47] empezó a desarrollar el sistema quirúrgico Da Vinci. Un robot formado por una consola ergonómica donde el cirujano puede operar sentado al paciente a través de unos brazos robóticos teleoperados. [48]

4.4.2. Clasificación por tipo de *Feedback*

Existen diferentes dispositivos hápticos en el mercado que podemos englobar dentro de dos grandes grupos:

- *Feedback* de fuerza:
  Nos aportan datos relacionados con la dureza y peso del objeto virtual, como por ejemplo los exoesqueletos.

- *Feedback* táctil:
  Nos dan información sobre la geometría de nuestro objeto virtual, su temperatura o rugosidad a través de los mecanorreceptores que se encuentran en la piel.

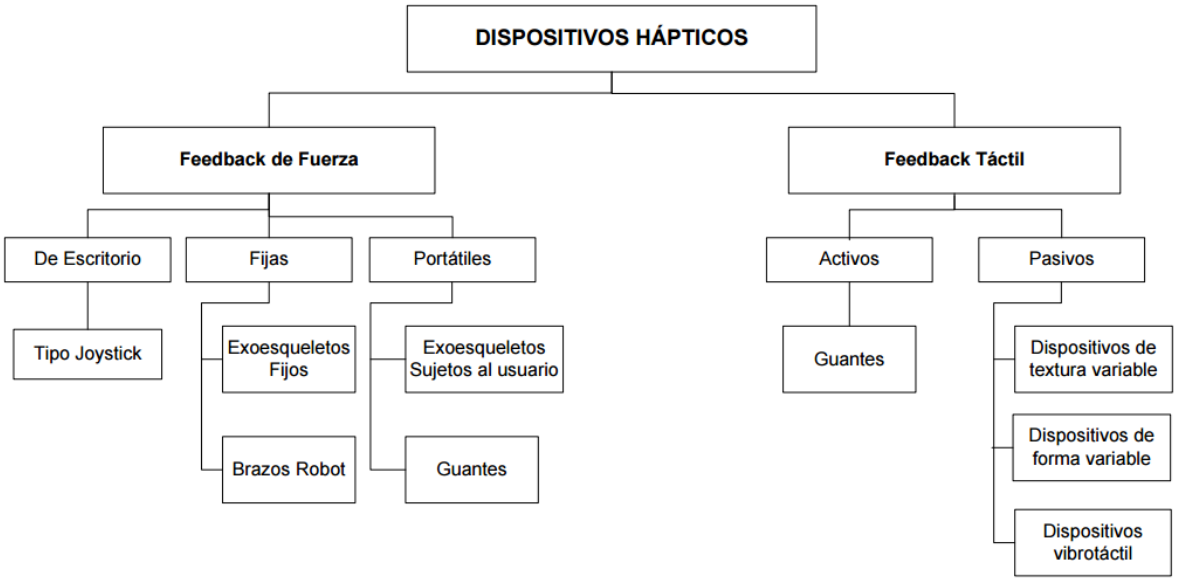


Fig. 4.6 Clasificación general de los dispositivos hápticos. (**Fuente:** Elaboración Propia)

4.4.2.1. *Feedback* de Fuerza

Los interfaces con *feedback* de fuerza son los que más se usan en la actualidad en las aplicaciones de realidad virtual. Con ellos conseguimos tocar y manipular objetos virtuales de forma realista, notando que el objeto ocupa realmente un volumen determinado en el espacio. Con estos interfaces podríamos establecer un plano virtual, y cuando el usuario lo toque y quiera traspasarlo, de alguna forma el sistema virtual se lo impida. Además con estos sistemas podemos simular características de los objetos que antes no podíamos tales como elasticidad, viscosidad, adherencia, etc. Dentro de este grupo podemos distinguir diferentes tipos de interfaces.

Fijas y portátiles. Exoesqueletos:

Son armazones colocados sobre algunas articulaciones y miembros del usuario que de forma controlada permiten aplicar una resistencia al movimiento y limitar los gestos del usuario. Para ello se utilizan sistemas neumáticos o eléctricos. Existen exoesqueletos que están sujetos a alguna estructura fija y otros que son “vestibles” sujetos simplemente al cuerpo del usuario. Los que están fijos permiten una mayor calidad en la simulación, ya que permiten simular objetos más grandes, simular mejor la dureza, etc. Por lo contrario no dan movilidad al usuario, por lo que el entorno virtual ha de estar confinado en un espacio relativamente pequeño. Por su propia naturaleza, con los exoesqueletos vestibles no se puede coger cualquier objeto, solo aquellos que están flotantes. Se pueden construir exoesqueletos para cualquier parte del cuerpo pero lo usual es que se adapten para las extremidades superiores.

En la actualidad uno de los guantes más comercializado es el Cybergrasp [49], fabricado por Immersion Co. El Cybergrasp consiste en una estructura exoesquelética fijada a la parte posterior de la mano, que es accionada por unos actuadores instalados fuera de ésta, en una caja de control, con el objetivo de facilitar su manejo aligerando su peso, de aproximadamente 450 gr. La fuerza máxima que puede aplicar sobre cada dedo es de 12N.

Fig. 4.7 Exoesqueleto CyberForce (**Fuente:** CyberGlove)

De escritorio (Tipo Joystick):

Este tipo de dispositivos basan su funcionamiento en no permitir al usuario tocar de forma directa el objeto virtual, sino que lo hacen a través de un medio físico intermedio, como un lápiz o dedal. De esta forma, el medio intermedio actúa como un filtro, que transmite al usuario de forma transparente la sensación cinestésica producida por el hecho de tocar el objeto, pero anula la sensación táctil. Las principales utilidades que tienen estos dispositivos suelen ser para el entrenamiento en cirugía, ya que son habilidades manuales muy precisas y costosas de realizar. Es típico ver sistemas de realimentación de fuerzas con instrumentos quirúrgicos acoplados en su terminal.

Posiblemente la gama de dispositivos hápticos más comercializada, conocida y utilizada sea la gama Phantom de la compañía Geomagic. Existen diferentes modelos de Phantom [50] que varían principalmente en los grados de libertad que ofrecen a la hora de actuar, su espacio de trabajo y la capacidad que disponen para resistir fuerzas.

4.4.2.2. *Feedback* táctil

Los dispositivos de *feedback* táctil nos aportan información sobre la geometría, temperatura o rugosidad de los objetos virtuales. Según Gibson [51], podemos diferenciar dos grandes grupos de percepción táctil, el tacto activo y el tacto pasivo. Entendemos por tacto activo, aquellas experiencias en las que el

usuario palpa activamente el objeto, mientras que el tacto pasivo es aquella información que el usuario recibe sin ser buscada intencionadamente.

El tacto activo implica un componente cinestésico añadido a la estimulación de la piel, por ejemplo, tenemos que mover los dedos para poder tocar un objeto con nuestra mano. Por el contrario, el tacto pasivo, suele darse en ausencia de cinestésia, ya que podemos ser tocados sin que nos movamos.

Dentro de este grupo podemos clasificarlos en función de los actuadores [52]:

Vibradores: Utilizan la vibración a una frecuencia determinada con el uso de solenoides que mediante la creación de un campo magnético son capaces de hacer vibrar la superficie que cubre dicho solenoide. O mediante el uso de motores eléctricos que con su rotación a una frecuencia determinada haga vibrar la piel sobre la que está colocado. Un ejemplo de este tipo de dispositivos es Cybertouch, un complemento para el guante CyberGlove [49] que provee realimentación táctil mediante estimuladores situados en las puntas de los dedos y en la palma de la mano.

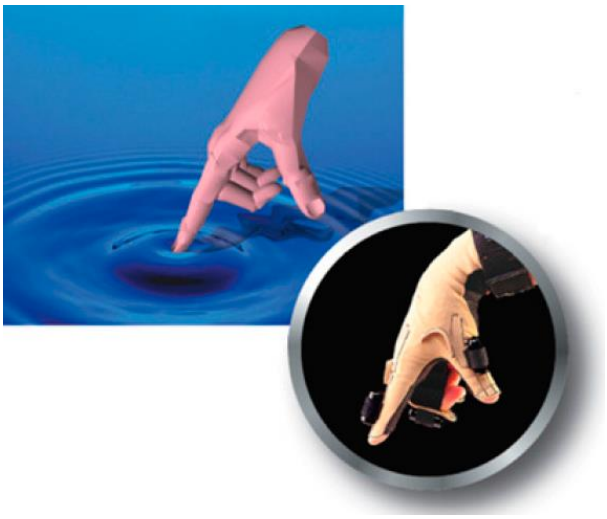


Fig. 4.8 Dispositivos Hápticos vibro-táctiles (**Fuente:** Cybertouch)

Neumáticos: Su función es producir una sensación táctil muy localizada, para ello se basa en sistemas de compresión de aire que se puedan inyectar hacia la piel del usuario de forma muy localizada. El principal problema de esta tecnología es la necesidad de obtener aire comprimido de forma fluida, con lo que el tamaño de los sistemas aumenta considerablemente.

Fig. 4.9 Dispositivos Hápticos Neumáticos. (**Fuente:** José San Martín)

Mecánicos: Estimulan la piel de forma directa, mediante actuadores mecánicos para lograr una estimulación táctil precisa. Su principal aplicación es la simulación de texturas, algo que no se podría realizar con las anteriores tecnologías debido a su falta de resolución espacial. Normalmente suelen estar formados por una matriz de pequeñas agujas que realizan presión sobre la piel del usuario. Aunque se suelen utilizar para estimular la yema de los dedos no se suelen añadir en los guantes de realidad virtual ya que para que las agujas funcionen correctamente necesitan mucho *hardware* asociado, haciendo que las dimensiones y peso del dispositivo sean muy grandes.

Electrocutáneos: Estos estimuladores se basan en la propiedad que tienen los mecanorreceptores de la piel de ser células nerviosas, con lo cual, ante una corriente eléctrica también producen estímulos similares a los que producen de forma normal cuando se activan bajo presión. Lo que se hace es pasar una pequeña corriente eléctrica, lo suficientemente pequeña para no producir dolor y lo suficientemente grande como para estimular las células mecanorreceptoras, por medio de una serie de electrodos colocados en la zona deseada. Así se consigue una sensación no muy localizada.

Térmicos: También podemos estimular los receptores térmicos de la piel, no tanto para simular la presión ejercida por un objeto virtual, sino para simular la naturaleza de la que está hecho el objeto. Para estimular a estos receptores se utilizan módulos termoeléctricos que son capaces de generar un gradiente térmico en función de la tensión aplicada de forma localizada sobre la piel del usuario. El problema de estos dispositivos es la gran inercia que tienen al cambio de temperatura, por lo que se utilizan disipadores que faciliten el control de la temperatura aplicada al usuario.

4.4.3. Principales dispositivos

A continuación detallaremos los dispositivos hápticos más avanzados en el mercado actual, siendo referentes en el mundo de la tecnología.

4.4.3.1. Dispositivos de escritorio

Los dispositivos de escritorio permiten una interacción puntual con los objetos virtuales a través de un lápiz de mano. Dentro de este grupo, Geomagic [50] es una de las empresas más importantes en este sector. Existen distintos dispositivos en función de los requisitos y necesidades. Estos productos, varían de tamaño en función del espacio de trabajo, resistencias, fuerzas, grados de libertad o precisión.

En la actualidad dispone de dos grandes grupos de dispositivos:

GeomagicTouch: Permiten una amplia gama de movimientos, equivalente al movimiento de la mano con giro muñeca. En general son dispositivos destinados a fines académicos, diseño gráfico o investigación.

GeomagicPhantom: Dispositivo de mayor tamaño que el tipo anterior, que permite movimientos equivalentes al movimiento del brazo con giro de codo o de hombro además del movimiento del giro de la muñeca. Este grupo, también está destinado con fines académicos, diseño gráfico o investigación.

Ambos pueden medir de forma precisa la posición espacial 3D a lo largo de los ejes (x, y, z) y la orientación (inclinación, giro y dirección) del lápiz de mano. Además de utilizan motores para crear el *feedback* de fuerza en la mano del usuario y poder simular el tacto y permitir interactuar con objetos virtuales.

Fig. 4.10 Geomagic Touch (**Fuente:** Geomagic)

Hay información más detallada acerca de los dispositivos Geomagic, en el anexo V.

4.4.3.2. Guantes hápticos

Existen diferentes fabricantes de guantes hápticos, pero si uno cabe destacar es CyberGlove [49], ya que dentro de su gama de productos disponen de diferentes guantes que responden a varios estímulos hápticos.

Entre ellos encontramos guantes que responden a estímulos táctiles como el CyberTouch, la forma como el CyberGrasp, y la fuerza, como el CyberForce.

CyberTouch. Es un guante de *feedback* táctil que incorpora pequeños estimuladores vibro-táctiles en cada dedo y en la palma de la mano. Cada estimulador puede programarse individualmente para sentir diferentes sensaciones de toque en cada dedo. Es posible, gracias al kit de desarrollo, diseñar diferentes patrones táctiles.

Fig. 4.11 CyberTouch (**Fuente**: Cyberglove).

CyberGrasp. Es un guante de *Feedback* de fuerza que permite coger objetos virtuales sintiendo su forma y tamaño dado que incorpora un exoesqueleto ligero que agrega una resistencia a cada dedo.

Fig. 4.12 CyberGrasp (**Fuente**: Cyberglove)

CyberForce. No es un guante como tal, sino que se trata de una extensión diseñada para trabajar junto con el exoesqueleto de CyberGrasp

proporcionando una retroalimentación de fuerza que además de la forma y tamaño permite sentir el peso de los objetos virtuales.

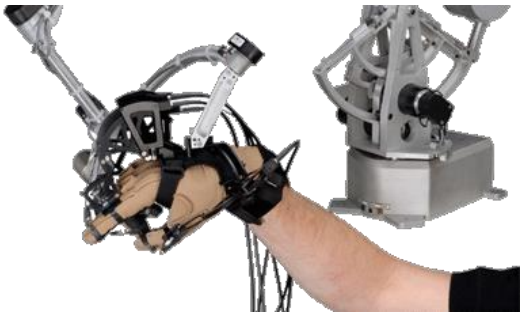


Fig. 4.13 CyberGrasp junto con CyberForce. (**Fuente**: CyberGlove)

Para más información acerca de los dispositivos se puede consultar el anexo VI con información acerca de sus características.

Es de esperar que para poder interactuar con el entorno virtual sea necesario disponer de la posición y captura de movimiento de mano y dedos. Por este motivo la compañía CyberGlove dispone del guante **MocapGlove** que permite registrar una movilidad física de forma precisa.

En función de la necesidad, podemos utilizar cada uno de los guantes comentados de forma independiente o de forma conjunta. Es decir, la unión de los cuatro tipos de guantes, hará posible, por ejemplo, mover objetos virtuales notando su textura, forma y sintiendo la fuerza que las leyes físicas ejercen sobre la mano.

Pese a que CyberGlove es un referente en el sector de guantes hápticos, cabe mencionar que durante el último año, empresas, como la española Neurodigital Technologies [53], están desarrollando guantes hápticos como GloveOne [54], siendo la primera empresa española que colabora con la plataforma OSVR [55] (Open-Source Virtual Reality), que es la plataforma de *software* que está estableciendo los estándares abiertos de la VR orientada a los videojuegos.

Fig. 4.14 Usuario de GloveOne utilizando Gear VR de Samsung [56] (**fuente**: GloveOne)

4.4.3.3. Force Touch

En la última década, el uso de smartphones y tablets ha aumentado considerablemente, añadiendo en cada nueva generación mejores procesadores, pantallas o cámaras entre otros. Es por ello que no es de extrañar que las empresas busquen incluir y mejorar la tecnología háptica de estos dispositivos. Durante los últimos meses, empresas como Apple o Microsoft han estado trabajando en este terreno.

Apple está mostrando tener un gran interés en tecnología háptica como se ha podido comprobar con el lanzamientos del Iphone 6s y su tecnología 3D Touch [57]. Esta tecnología permite detectar la presión ejercida sobre la pantalla recibiendo diferentes respuestas hápticas en función de la presión que se ejerce. Además, la empresa ha patentado un nuevo sistema que permitirá simular texturas y cambios de temperatura en pantallas táctiles [58].

Como se ha comentado, otras compañías como Microsoft [43] o Disney Research [59] están desarrollando sistemas de retroalimentación háptica, que permitirán simular texturas en superficies lisas.

CAPÍTULO 5. DISEÑO Y SELECCIÓN

5.1. Diseño

Una vez adquirido cierto conocimiento sobre dispositivos de entrada y salida, entornos virtuales, aumentados y sistemas hápticos, en el presente capítulo abordaremos las bases para el diseño de un sistema de entrada y salida háptico.

En la figura 5.1 [60] se puede observar un sistema de VR con respuesta visual, auditiva y táctil. En dicho sistema, se aprecia como el flujo de información visual y auditiva sólo viaja en un sentido. El usuario puede escuchar música a través de unos altavoces y ver imágenes a través de una pantalla pero no puede enviar información a través de estos dispositivos. En cambio, un dispositivo háptico permite intercambiar información de forma bidireccional ya que el usuario proporciona una entrada al entorno virtual que provoca cambios visuales y de desplazamiento que como consecuencia proporcionará una salida hacia el usuario mediante la actuación del tacto.

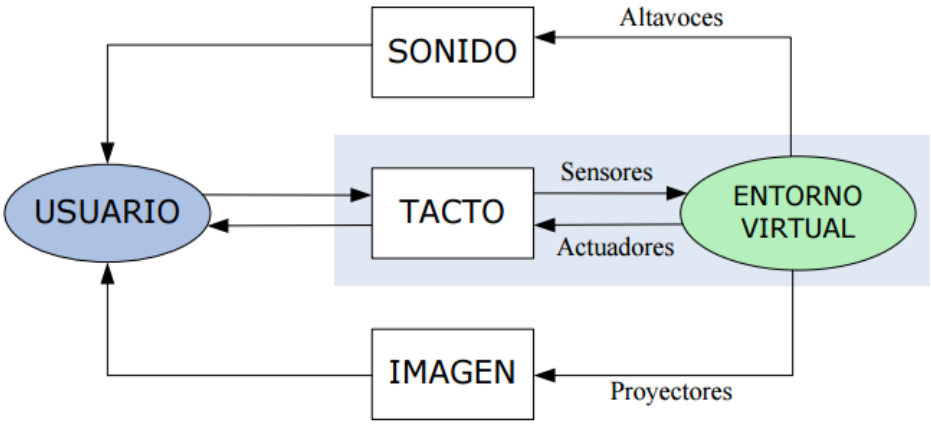


Fig. 5.1 Flujo general de un sistema háptico multisensorial (**Fuente:** Josune Hernantes)

Hemos adaptado la arquitectura general de un sistema háptico multisensorial para poder llevar a cabo nuestro proyecto y así poder definir las diferentes capas que formaran parte del trabajo y para poder determinar las fases de su creación.

Esta arquitectura consta de tres grandes capas como se puede ver en la figura 5.2:

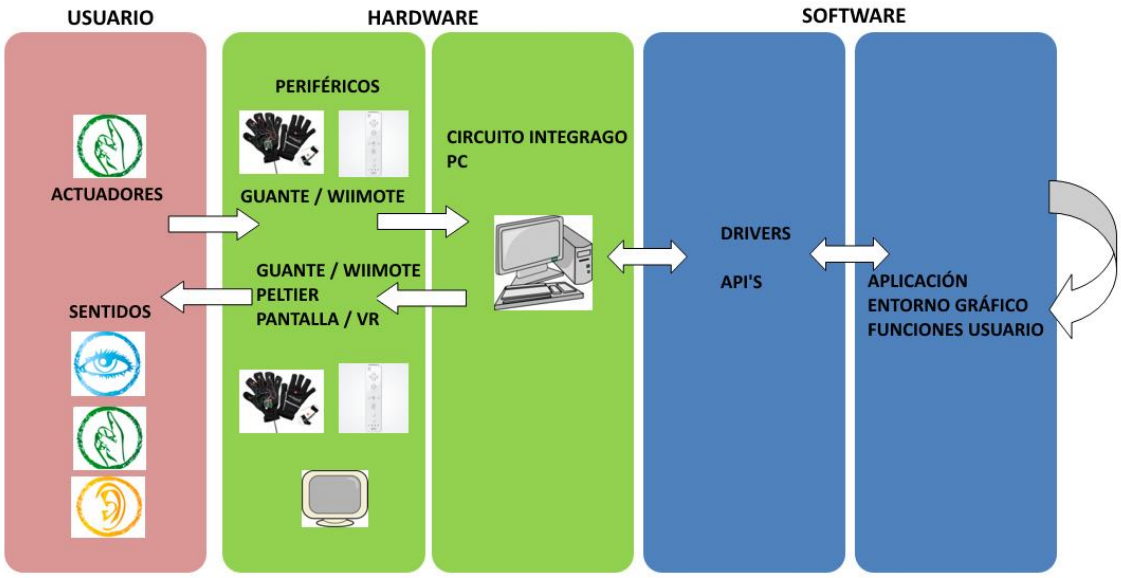


Fig. 5.2 Arquitectura concreta de nuestro diseño HHCI (**Fuente:** Elaboración propia basada en Wikipedia)

Capa Usuario: El usuario interactúa dentro del sistema de forma bidireccional. Puede enviar y recibir información a través de la capa *hardware*. Es decir, mediante nuestros "actuadores naturales" enviamos una orden a través del dispositivo de entrada que será procesada para volver en forma de respuesta a través de un dispositivo de salida hacia cualquiera de nuestros "sentidos".

Capa *Hardware*: En esta capa podemos diferenciar dos subcapas. Una primera capa formada por dispositivos de entrada y salida. Y una segunda capa que hace de intermediario entre los dispositivos de entrada y salida y el *software* procesando la información. Tal y como se observa en la figura 5.2 es en la primera capa de *hardware* donde se encuentra el dispositivo háptico de nuestro diseño.

Capa *Software*: Por un lado tenemos una subcapa con todo lo relacionado con la interconexión con el *Hardware*: (API, drivers, etc.) que hace que podamos utilizar y aprovechar los componentes. Por otro lado, tenemos la programación de las funciones de usuario, es decir, el desarrollo de la aplicación donde se definen las instrucciones y eventos.

El renderizado es una fase del diseño muy importante dentro de la capa de *software* a la hora de crear el sistema háptico dado que es un proceso que se encarga de representar el entorno virtual al usuario mediante determinados estímulos [61].

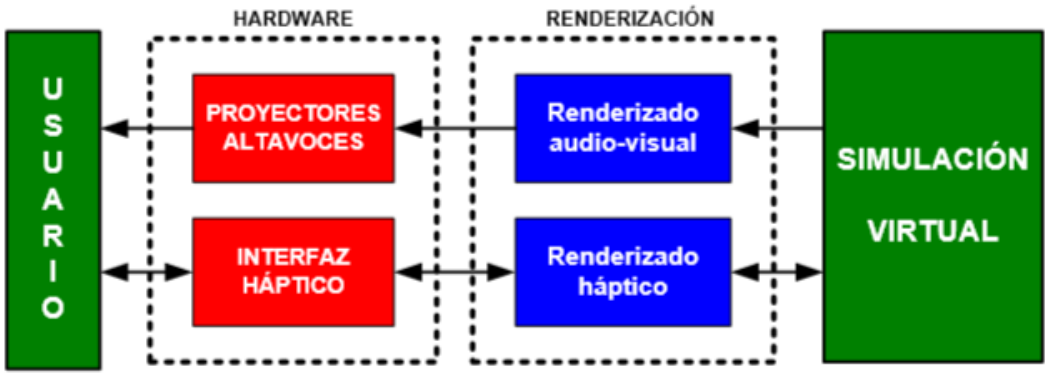


Fig. 5.3 Esquema diseño dispositivo háptico (**Fuente:** Iñaki Diaz)

Como podemos ver en las figura 5.2 y 5.3, la información no solo va en una dirección, sino que es bidireccional. Es decir, enviamos una señal a través de un componente (mando o un guante con tecnología háptica) y recibimos un estímulo a través de otro componente.

5.2. Grados de libertad

Los grados de libertad, también llamados DoF (Degree of Freedom) hacen referencia al número de movimientos independientes que un cuerpo puede realizar. Es decir, un grado de libertad es la capacidad de moverse a lo largo de un eje (movimiento lineal) o de rotar a lo largo de un eje (movimiento rotacional). El número de grados de libertad, coincide con el número de ecuaciones necesarias para describir el movimiento, es decir, a más grados de libertad, más preciso será el movimiento de nuestro dispositivo.

Para escenarios en dos dimensiones hablamos de tres grados de libertad y en escenarios en tres dimensiones hablamos en seis grados de libertad.

En la siguiente imagen, podemos ver los seis grados de libertad: adelante/atrás (forward/back), arriba/abajo (up/down), izquierda/derecha (left/right), cabecear (pitch), guiñar (yaw), rodar (roll) [62]

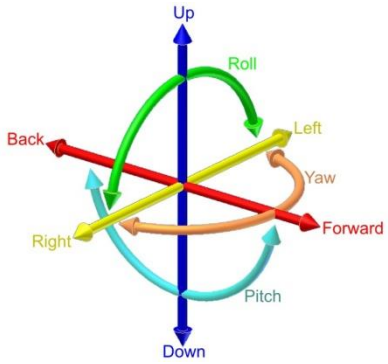


Fig. 5.4 Grados de Libertad (**Fuente:** Wikipedia)

5.3. Renderizado háptico y Colisiones

Cuando se habla de diseños hápticos es frecuente oír hablar de renderizado. Renderizado (o Render) es el término que se utiliza para referirse al proceso de generar imágenes a partir de un modelo y está estrechamente ligado con la háptica y la generación de entornos virtuales [63].

Podemos diferenciar entre dos grandes tipos de renderizado, el renderizado gráfico y responsable por ejemplo de que el usuario pueda mover una cámara dentro del mundo virtual y el renderizado háptico.

El renderizado háptico tiene como objetivo la transmisión de sensaciones táctiles que permitan estimular el sentido del tacto para conseguir que las simulaciones virtuales sean más realistas a la vez de permitir mayor grado de interacción. La utilización de algoritmos de renderizado háptico permite calcular las fuerzas necesarias que harán que el *feedback* transmitido a través del dispositivo háptico sea lo más realista posible.

Dichos algoritmos se componen de [60]:

- Detección de colisiones
- Cálculo de la respuesta a la colisión
- Módulo de control

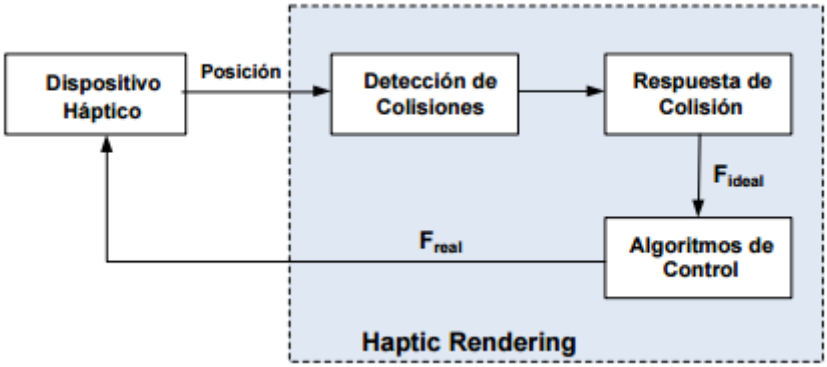


Fig. 5.5 Renderizado háptico. (**Fuente**: Josune Hernantes)

5.3.1. Detección de colisiones

Este módulo es el encargado de detectar la posición del dispositivo háptico e información del entorno virtual para poder detectar si existe o no colisión.

5.3.2. Cálculo de la respuesta de colisión

Este módulo se encarga de tomar la información acerca de la colisión producida y calcular la "Fuerza Ideal" de interacción entre el objeto virtual y su entorno.

5.3.3. Módulo de control

Los algoritmos de módulo de control son los responsables de devolver al usuario, a través del dispositivo háptico, una respuesta con "Fuerza Real" lo más aproximada posible a la "Fuerza Ideal" calculada en la respuesta de colisión.

5.4. Selección del dispositivo de entrada y salida

La idea inicial de desarrollo fue la de replicar en las instalaciones de la CIMNE y EETAC el proyecto de guante háptico que se realizó en la Universidad de Cornell [64]. Nuestra implementación se basaba en agregar *feedback* háptico térmico, poder sentir la fuerza que ejerce un objeto virtual en nuestro guante y además que poder actuar como periférico de entrada. Sabíamos que era una implementación ambiciosa y debido al corto plazo de tiempo que disponíamos decidimos sacrificar la implemetación de *feedback* de fuerza.

Tras comprobar el diseño visto en los apartados anteriores, empezamos a evaluar la viabilidad de la implementación. Nos encontramos con limitaciones de los actuadores para generar la sensación de frío y calor, dado que el peltier (actuador que habíamos considerado), al estar un tiempo actuando como

fuente de calor, no daba tiempo suficiente a enfriarse para generar frío en el caso que fuera necesario, por lo que el dispositivo siempre estaría caliente. El diseño lo teníamos claro, pero nuevamente, estábamos delimitados por el corto plazo de entrega que disponíamos.

Después de comprobar que esta implementación no era viable, decidimos cambiar de dispositivo. Utilizar algún instrumento que permitiera extrapolar la idea inicial, pero que no llevara tanto tiempo de desarrollo. La solución fue utilizar el mando WiiMote de Nintendo®. Cambiamos el *feedback* térmico por una respuesta vibro-táctil.

¿Por qué hemos escogido el controlador de la Wii para nuestro experimento? El WiiMote contiene uno de los elementos necesarios para describir la importancia de la háptica en diferentes situaciones. Este componente es la función de vibración, gracias a este actuador, podemos recibir *feedback* de un elemento virtual.

El controlador de Wii, además de incorporar actuadores vibratorios también dispone de unos sensores de movimiento y acelerómetros que permiten ubicar el controlador en el espacio y nos permitirán en nuestro caso poder interactuar con nuestras aplicaciones sin necesidad de cables.

En el caso de nuestro diseño, el mando WiiMote dispone de 6 grados de libertad, dado que su utilización es para entornos en tres dimensiones. A pesar de ello y dado que nuestras aplicaciones se han desarrollado en entornos de dos dimensiones, solo utilizamos un grado de libertad para el teclado numérico, porque sólo permitimos el movimiento en uno de los ejes (derecha e izquierda) y dos grados de libertad para el simulador, dado que permitimos al usuario moverse en dos de los ejes (derecha, izquierda, delante y detrás).

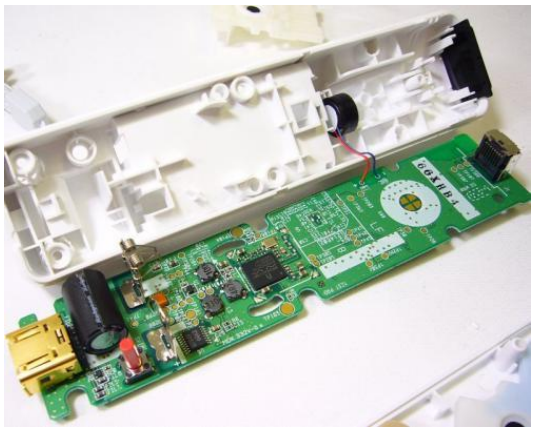


Fig. 5.6 Controlador WiiMote. Componentes (**Fuente:** Sparkfun)

El acelerómetro, característica que hemos utilizado en nuestros desarrollos, dispone de información acerca de la aceleración del movimiento del control en sus tres ejes:

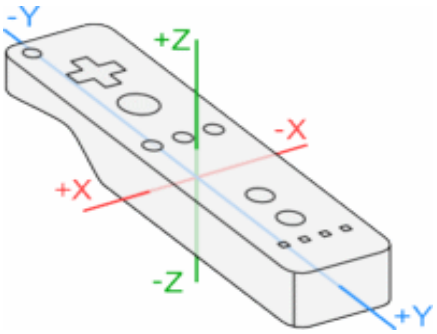


Fig. 5.7 Ejes Acelerómetro WiiMote (**Fuente:** wiimotelib.codeplex.com**)**

Para poder recoger los valores del acelerómetro o para poder activar la función vibración es necesario añadir a la solución la librería WiimoteLib.dll. Una vez instalada, y reconocida por nuestro entorno de programación empezamos con el desarrollo de la aplicación. En nuestro caso hemos desarrollado dos aplicaciones, ambas con la posibilidad de activar y desactivar la tecnología háptica y así permitir al propio usuario notar la sensación e importancia que tiene una vibración en el lugar y momento adecuado.

CAPÍTULO 6. IMPLEMENTACIÓN

6.1. Lenguaje y entorno de programación

Como hemos visto en el diseño, la siguiente parte fundamental es la capa de *software*. Para poder realizar nuestro proyecto hemos decidido utilizar el lenguaje de programación C Sharp, creando una solución Windows Forms. La librería utilizada para poder utilizar las funciones que nos permiten interactuar con el mando está disponible para dicho lenguaje o Visual Basic. Dicha librería fue creada por Brian Peek [65], desarrollador de Microsoft y autor de numerosos artículos y proyectos. Dado que nosotros teníamos cierta experiencia con C#, decidimos empezar nuestro proyecto utilizando este lenguaje de programación.

El entorno de programación utilizado ha sido Sharp Develop [66], por ser un programa de código abierto y no necesitar ninguna licencia para poder ser utilizado.

Fig. 6.1 Entorno de programación c# (**Fuente:** Elaboración propia)

6.2. Requisitos

Es recomendable disponer de un ordenador con Windows, en este caso hemos utilizado Windows 8.1 por estar más familiarizados con este entorno y tener instalada la última versión de Sharp Develop. También es necesario un receptor de Bluetooth en el ordenador y el controlador WiiMote de Nintendo.

Al conectar por primera vez el WiiMote con el ordenador a través del sistema operativo, este instala los drivers necesarios para detectar que es el mando de Nintendo®. Realizando las pruebas nos encontramos con un problema y es que cada vez que reiniciábamos el sistema o se acababa la batería del controlador era necesario eliminar el dispositivo y volver a instalarlo y emparejarlo. Hemos buscado información acerca de la incidencia, pero las soluciones ofrecidas por Microsoft no nos ayudaron a esclarecer lo sucedido, con lo que siempre hemos tenido que recurrir al mismo método para hacer funcionar el Wiimote.

6.3. Implementación de las soluciones

Para la implementación de la solución se han escogido dos aplicaciones. La primera de ellas, un teclado virtual númerico, fácil de usar y en el que la respuesta háptica, en forma de vibración, no tenga una connotación negativa sino informativa. Y una segunda aplicación, en forma de simulador de conducción, donde la háptica actuará a modo de advertencia.

Ambas aplicaciones se han desarrollado en paralelo. En el teclado virtual el usuario puede moverse por la interfaz gracias al acelerómetro, recibiendo un feedback háptico vibratorio cada vez que se cambia el foco de un botón o se presiona uno de ellos.

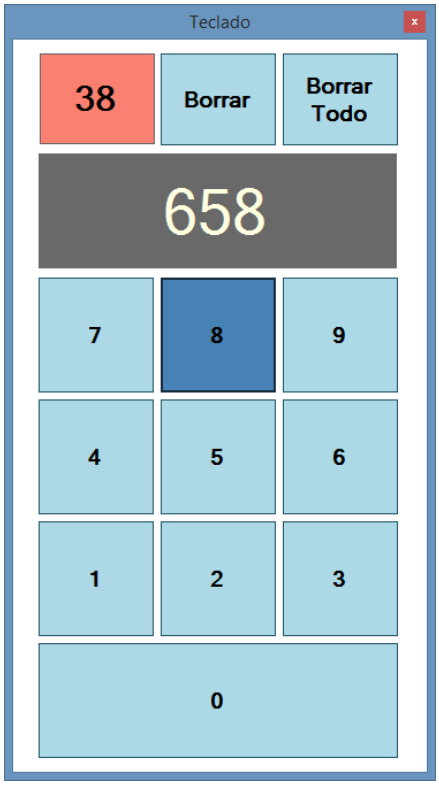


Fig. 6.2 Aplicación Teclado Virtual (**Fuente:** Elaboración propia)

La segunda aplicación consiste en un simulador de conducción de vista aérea en el que el usuario puede avanzar por una carretera de dos carriles. El vehículo puede moverse hacia la derecha o izquierda gracias al acelerómetro. Para esta aplicación se recibirá una leve vibración cuando se cambie de carril y una vibración más intensa cuando se colisione con algún objeto o el vehículo se sitúe en los arcenes de los laterales de la carretera.

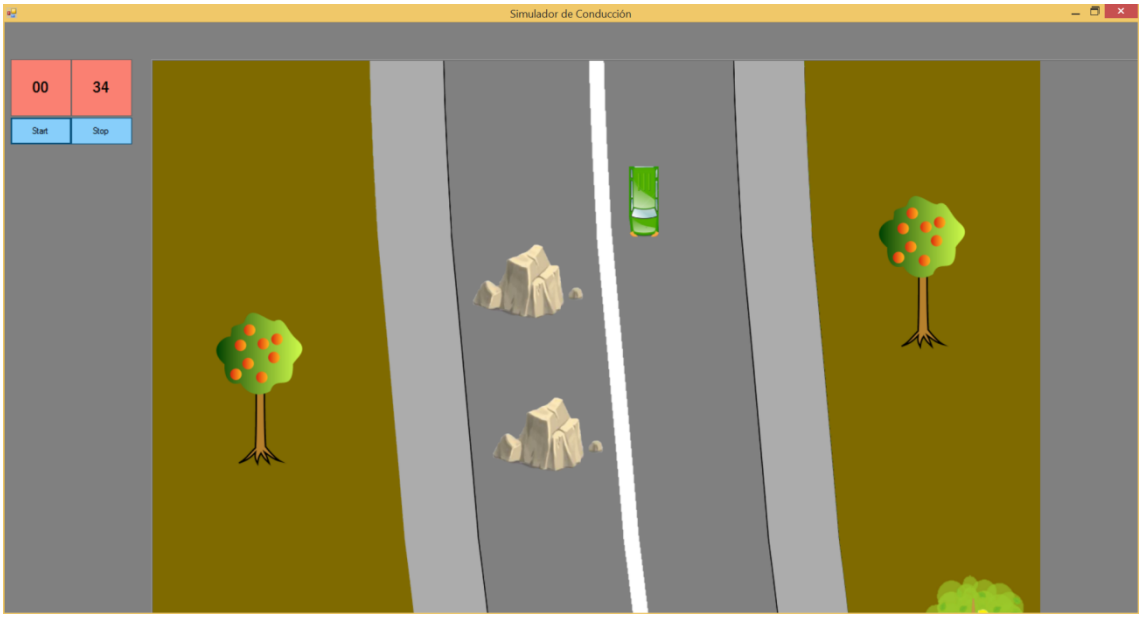


Fig. 6.3 Aplicación Simulador de Conducción (**Fuente**: Elaboración propia)

Ambas aplicaciones comparten las llamadas a los métodos proporcionados por la librería WiimoteLib.dll que se describirá en el siguiente apartado.

6.4. Funcionamiento básico de la librería WiimoteLib.dll

Como hemos comentado en el apartado anterior, las aplicaciones comparten código fuente tanto para la programación del método de entrada como la del método de salida. Las llamadas a los diferentes métodos de la librería son esenciales para poder conectar el controlador, para poder hacer uso del acelerómetro y sobre todo para poder utilizar la vibración del mando y poder comprobar la utilidad que puede llegar a tener un sistema háptico dentro de un dispositivo de entrada y salida (para mas información acerca de la librería consultar Anexo IX).

6.4.1. Instalación y configuración de la librería

Uno de los requisitos para poder implementar nuestra solución, es la utilización de la librería Wiimotelib.dll. Para ello, descargamos de CodePlex [65] la última versión de la librería, v1.7. Una vez descargada se añade la referencia al proyecto incluyendo el código fuente al espacio de nombres:

```
using WiimoteLib;
```

Para poder acceder a la información de la librería es necesario referenciarla:

```
private Wiimote wm;
```

Y crear una instancia:

```
try
{
    wm = new Wiimote();
}
catch (System.IO.IOException ioe)
{
    Console.WriteLine("No es posible ejecutar la aplicación: " + ioe.Message);
}
```

Para que la solución funcione correctamente será necesario que el dispositivo esté emparejado con el equipo.

6.4.2. Conexión del WiiMote

Una vez creada la instancia, se deben registrar los eventos de control del mando de Wii. Para ello existen dos tipos de eventos, *WiimoteExtensionChanged* que es generado por la conexión y desconexión de extensiones (Nunchuck) y *WiimoteChanged*, generado por el propio controlador cuando el mando ha cambiado de estado (presionar un botón o posición del acelerómetro entre otros). [67]

Dado que no vamos a usar extensiones solo es necesario incluir el evento *WiimoteChanged* al inicio del programa.

```
wm.WiimoteChanged += new System.EventHandler<WiimoteChangedEventArgs>(wm_WiimoteChanged);
```

Una vez establecido el registro de eventos, debemos establecer la conexión con el mando. Como la conexión del Wiimote y el sistema operativo Windows 8.1 no es fiable, es aconsejable proporcionar un control de errores:

```
try
{
   wm.Connect();
}

catch (WiimoteNotFoundException e)

{

    MessageBox.Show(e.Message);

}
```

6.4.3. WiiMote como dispositivo de entrada

Es necesario establecer el nivel de reporte, es decir, determinar el tipo de información que deseamos obtener del WiiMote como método de entrada. Podemos elegir entre las siguientes constantes definidas en *InputReport*:

- **Status**. Estado general del WiiMote.
- **ReadData**. Datos de memoria interna.
- **Buttons**. Botones pulsados.
- **ButtonsAccel**. Botones pulsados y acelerómetro
- **IRAccel**. Botones, acelerómetro e infra-rojos.
- **ButtonsExtension**. Botones pulsados y Extensiones
- **ExtensionAccel**. Botones pulsados, acelerómetro y extensiones.
- **IRExtensioAccel**. Botones pulsados, Acelerómetro, infra-rojos y extensiones.

Para nuestros desarrollos necesitamos información del acelerómetro y botones pulsados. Inicialmente se planteó la idea de utilizar la información obtenida a través del infra-rojo, pero dada la poca precisión del receptor de infra-rojos del mando no fue posible controlar de forma adecuada las aplicaciones.

El código necesario para poder establecer el informe es el siguiente:

```
wm.SetReportType(InputReport.IRAccel, true);
```

La utilización de los eventos ante los cambios del Wiimote es una de las partes más importantes para poder lograr el manejo del controlador como dispositivo de entrada, es decir, indicar las acciones que deseamos realizar al pulsar un botón o mover el mando.

Para ello es necesario utilizar el método wm_WiimoteChanged, éste es invocado cuando ocurre un cambio en las variables de control antes mencionadas. Como se muestra en el código siguiente, cuando el botón “+” cambia a verdadero (botón pulsado), la variable rumb (vibración) pasa a ser verdadera. Por el contrario, cuando el que el botón "-" cambia de estado a verdadero (botón encendido), la variable pasa a ser falsa.

```
void wm_WiimoteChanged(object sender, WiimoteChangedEventArgs e)

{//Funcíon que se activa cuando ocurre un cambio en el estado del control de wii
    if (e.WiimoteState.ButtonState.Plus == true)
    {
        rumb= true;
    }

    if (e.WiimoteState.ButtonState.Minus == true)
    {
        rumb = false;

    }
```

A continuación enumeramos las diferentes propiedades disponibles en la librería y que han sido utilizadas en alguna parte de nuestras implementaciones:

- **ButtonState.** Estado de los botones.
- **AccelState**. Estado del acelerómetro.

6.4.3.1. ButtonState

Como se ha explicado en el ejemplo anterior, la forma de acceder a cualquier botón se realiza a través de la comprobación del estado del mando:

```
e.WiimoteState.ButtonState.Right
```

En el caso de los botones, estas propiedades son booleanas, por lo tanto, podremos controlar con un “falso” si el botón no está presionado, o con un “verdadero” si el botón se encuentra presionado. Podemos encontrar los siguientes botones:

Botones superiores:
- ButtonState.A
- ButtonState.B

Botones medios:
- ButtonState.Plus (+)
- ButtonState.Minus (-)
- ButtonState.Home (Casa)

Botones inferiores:
- ButtonState.One
- ButtonState.Two

Control de dirección:
- ButtonState.Up
- ButtonState.Down
- ButtonState.Right
- ButtonState.Left

6.4.3.2. AccelState

Como su nombre indica, la propiedad AccelState, provee información acerca de la aceleración del movimiento del mando en sus tres ejes (ver figura 5.6). El rango de valores que utiliza se encuentra entre -1 y 1 y es del tipo *float.*

```
e.WiimoteState.AccelState.Values.Y
```

6.4.4. WiiMote como dispositivo de salida

Como hemos comentado anteriormente, el WiiMote, además de ser un dispositivo de entrada (como un controlador convencional), también actúa como dispositivo de salida. El dispositivo dispone de varios actuadores visuales como son los leds que incorpora en la parte inferior del mando y un zumbador, que permite la vibración del dispositivo.

6.4.4.1. SetLEDs

El WiiMote dispone de cuatro leds de color azul que nos permiten conocer el número de jugador de los cuatro disponibles. Esta propiedad es booleana, y el funcionamiento es similar al de los botones, por ello sólo permite disponer de dos estados “verdadero” cuando está encendido, y “falso” cuando está apagado. Siendo, cada uno de los valores “false”, las cuatro posiciones de los leds tal y como se muestra en la siguiente figura:

```
wm.SetLEDs(false, false, false, false);
```

6.4.4.2. SetRumble

Por último hay que establecer la parte más importante de nuestro proyecto. La vibración del WiiMote, que nos aportará el *feedback* háptico. El primer problema con el que nos encontramos fue la imposibilidad de determinar niveles de vibración, ya que esta propiedad es del tipo booleana. Por lo tanto, sólo tenemos la opción de hacer, o no, vibrar el dispositivo.

```
wm.SetRumble(false);
```

A pesar de no poder provocar diferentes intensidades de vibración con la librería, pudimos encontrar la manera de poder aplicar un tiempo de vibración en función del tipo de colisión. Es por ello que para vibraciones que necesitábamos que fueran más fuertes, optamos por dejar más tiempo la vibración encendida que para vibraciones con menor intensidad.

```
for (int i =0; i<10;i++)
{
wm.SetRumble(rumb);
}
wm.SetRumble(false);
```

6.4.5. Test de las funciones más importantes.

Previamente al desarrollo de las dos soluciones, comprobamos el funcionamiento de la librería gracias a una aplicación de ejemplo como se muestra en la figura 6.4 [65].

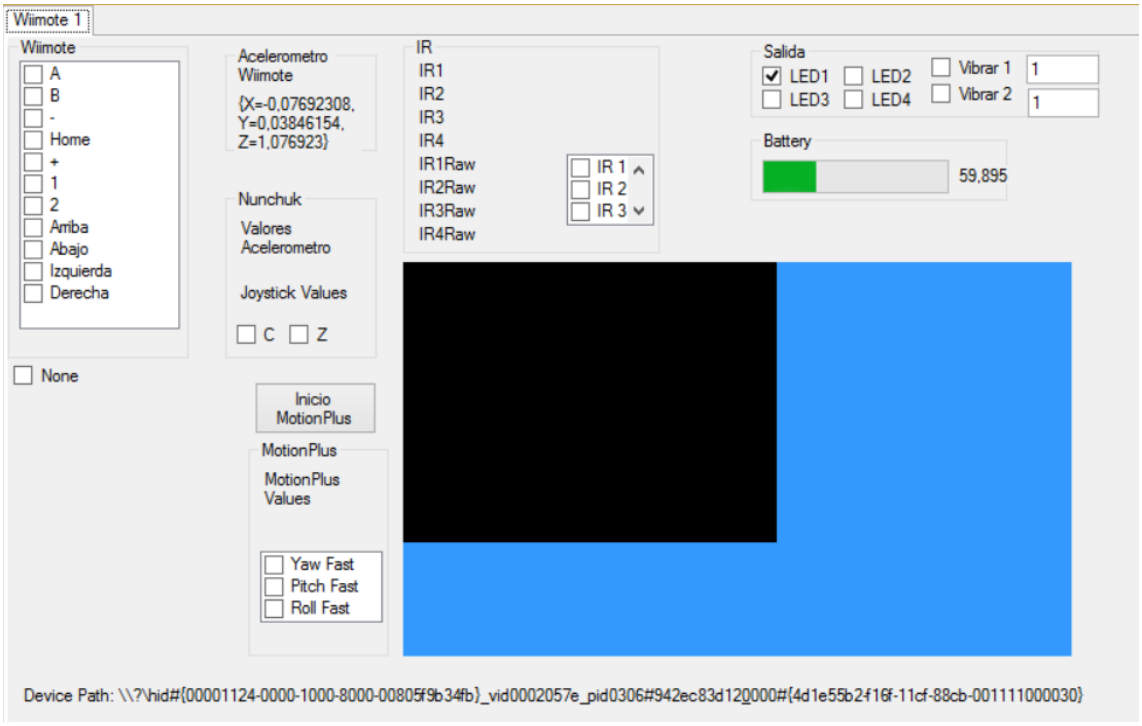


Fig. 6.4 Escenario de test. Comprobación de funcionamiento de la librería. (**Fuente:** Elaboración propia)

6.4.6. Problemas con el acceso entre hilos de C#

Durante la implementación de las aplicaciones nos encontramos con un problema del lenguaje C#. Cuando se ejecuta el código de la librería WiimoteLib por seguridad se ejecuta en un hilo diferente al de nuestra aplicación. Así que, cuando intentábamos realizar cualquier actualización de nuestros formularios a partir de un valor obtenido del WiiMote provocaba una excepción tal y como se muestra en la figura 6.5.

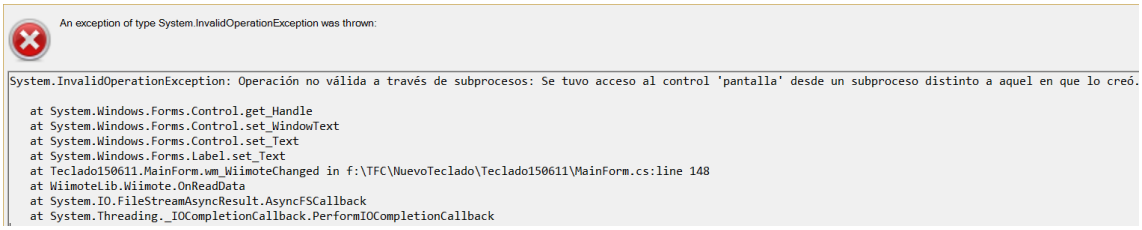


Fig. 6.5 Excepción: *System.InvalidOperationException: Operación no válida a través de subprocesos: Se tuvo acceso al control 'pantalla' desde un subproceso distinto a aquel en que lo creó.* **(Fuente:** Elaboración propia)

Para poder ejecutar las instrucciones que requerían accesos entre hilos distintos (actualizar la pantalla a través de los botones del WiiMote, presionar botones de la aplicación con los botones del mando, etc.) tuvimos que implementar nuevos métodos y llamarlos a través de delegados para poder invocarlos a través de las funciones de la librería. [68]

Por ejemplo, para poder presionar un botón de nuestra aplicación a través de un control del mando (botón A), deberemos programar el método que llamaremos a través del delegado:

```
//Apretar boton con mando

delegate void apretarbotondelegate (Button Bot);

public void apretarboton (Button Bot)
{

            Bot.PerformClick();

}
```

Y la invocación la realizaremos dentro del evento de cambios del Wiimote:

```
if (args.WiimoteState.ButtonState.A)
{

    if (InvokeRequired && botonapretado == true)
    {
        BeginInvoke(new apretarbotondelegate(apretarboton),
                    new object[]{
                        this.ActiveControl
                    });
```

6.5. Desarrollo del Teclado Virtual háptico

Como se ha comentado, se ha desarrollado un Teclado Virtual que emite una respuesta háptica en forma de vibración sobre el WiiMote cada vez que presionamos o nos desplazamos sobre sus teclas.

En los siguientes subapartados se detallarán las soluciones de implementación que se han escogido para resolver problemas tales como pulsar y desplazarnos por los botones de la aplicación a través de movimientos gestuales realizados con el mando (para mas información sobre el código fuente de la aplicación, consultar Anexo VI).

6.5.1. Desplazamiento

En la primera versión del teclado virtual, se optó por programar la aplicación conectando el mando pero interactuando tan solo con el ratón y teclado de nuestro ordenador. Dicha aplicación sólo emitía respuesta háptica, visual y auditiva al presionar los botones virtuales. Tal i como se ha comentado en el punto 6.5.4.2, se decidió programar una función que hacía vibrar el mando con una duración determinada.

```
void Button0Click(object sender, EventArgs e)
{
    pantalla.Text=TextEscrit + 0;
    TextEscrit = pantalla.Text;
    string path = @"C:\Users\Usuario\Desktop\wav-numero100\0.WAV";
    playSound(path);

    for (int i=0 ; i<=15;i++)
    {
    wm.SetRumble(true);
    }
    wm.WiimoteState.Rumble=false;

}
```

En la segunda versión de la aplicación, para simplificar la programación, se optó por realizar el desplazamiento sobre los botones (a partir de ahora foco) mediante la utilización del control de dirección (“cruceta del mando”).

```
if (args.WiimoteState.ButtonState.Left)
{
   if (InvokeRequired)
   {
      BeginInvoke(new updateBotonDelegate(updateBoton),
                  new object[]{
                      this.GetNextControl(ActiveControl, false)
                  });
                  }

   }
```

A pesar de su correcto funcionamiento, se observó que en esta implementación la háptica no aportaba valor añadido, ya que al presionar el control de dirección ya recibimos información de forma mecánica. Es decir, sabemos que hemos presionado el control porque nuestros dedos han notado la presión ejercida. Por lo tanto, añadir un *feedback* háptico en forma de vibración, no aportaba ninguna mejora sobre el teclado virtual.

En la versión actual, el desplazamiento se realiza de derecha a izquierda mediante el uso del acelerómetro del mando en el eje de coordenadas Y.

```
//Movemos el mando hacia la izquierda

if (args.WiimoteState.AccelState.Values.Y*Aceleracion == 1)
{

    if (ActiveControl.TabIndex != 0){
    if (InvokeRequired)
    {
        BeginInvoke(new updateBotonDelegate(updateBoton),
                    new object[]{
                        this.GetNextControl(ActiveControl, false)
                    });

        // Volvemos a poner el teclado con el color original
        colorboton();

    }

    Aceleracion=10;

    }
}
```

Con esta versión, cada vez que cambiamos el foco del botón recibimos una respuesta háptica y visual que nos permite conocer la posición del control actual.

6.5.2. Pulsación de botones

Desde la primera versión se decidió que cada vez que se presionara un botón se recibirían diferentes respuestas:

- ***Feedback* háptico**: Se emite una pequeña vibración cada vez que se presiona uno de los botones virtuales.
- ***Feedback* visual**: Se muestra por pantalla el valor del número presionado y la posición del foco actual.
- ***Feedback* auditivo**: Se emite la transcripción fonética del número presionado.

```
if (args.WiimoteState.ButtonState.A)
{

    if (InvokeRequired && botonapretado == true)
    {
        BeginInvoke(new apretarbotondelegate(apretarboton),
                    new object[]{
                        this.ActiveControl
                    });

        BeginInvoke(new updatePantallaInformacionDelegate(updatePantallaInformacion),
             new object[] {
                        this.pantalla, this.ActiveControl
                    });

    }
```

Durante esta implementación, el principal problema con el que nos encontramos fue controlar el tiempo que se mantenía presionado el botón del mando. Este problema causaba que por pantalla se mostrara de forma visual el dígito pulsado repetidas veces (dependiendo del tiempo que se mantenía pulsado el botón). Para solventarlo, la solución adoptada fue crear una variable booleana (*botonapretado*) que adquiere el valor de falso la primera vez que se muestra por pantalla el dígito y no cambia de valor hasta que cambia el foco.

6.5.3. Cronometro

En la última versión de la aplicación se ha incluido un cronometro implementado con la función *timer* que mide el tiempo en segundos y nos permitirá cuantificar futuros ensayos.

```
//Timer de control de crono
void Timer1Tick(object sender, EventArgs e)
{
    contador++;

    if (contador > 0 && contador <= 9)
    {
        crono.Text="0"+contador.ToString();
    }

    else
        crono.Text=contador.ToString();

}
```

6.5.4. Función háptica

Para poder valorar como contribuye la tecnología háptica en nuestra aplicación, se ha programado una función que permite habilitar o deshabilitar la vibración del mando.

```
//Activar opcion haptica pulsando el botón +
if (args.WiimoteState.ButtonState.Plus)
{

    rumb = true;

}

//Quitar opción haptica pulsando botón -
if (args.WiimoteState.ButtonState.Minus)
{

    rumb = false;

}
```

6.6. Desarrollo de un Simulador de Conducción

Se ha desarrollado un Simulador de Conducción que permite desplazar el vehículo lateralmente con el acelerómetro del mando WiiMote. En el simulador deberemos esquivar los obstáculos presentes en la carretera y evitar salirnos de los arcenes laterales (para mas información acerca del código fuente de la aplicación, consultar Anexo VIII).

Para el desarrollo de la aplicación hemos diferenciado varios subapartados clasificándolos de la siguiente manera:

6.6.1. Desplazamiento

En la primera versión de la aplicación se implementó una carretera sin curvas y con obstáculos donde el desplazamiento del vehículo podía realizarse con teclado, control de dirección y acelerómetro pudiendo recibir una respuesta visual y háptica en forma de vibración al chocarse con un obstáculo o saliéndose de la carretera.

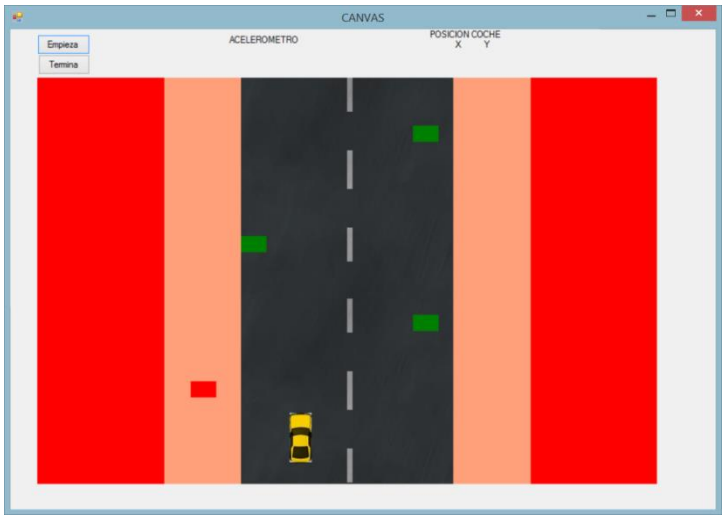


Fig. 6.6 Versión 1 Simulador de Conducción (**Fuente:** Elaboración propia)

En el desplazamiento lateral con el control de dirección del mando, a diferencia de la aplicación del Teclado Virtual, la háptica sí nos aporta valor añadido ya que permite corregir la trayectoria en el caso de colisionar con un obstáculo. En este escenario el manejo resultaba sencillo, por lo que finalmente optamos por desplazar el vehículo con el acelerómetro del mando de la Wii. Cuando inclinamos el mando hacia la derecha o izquierda, la posición del vehículo queda desplazada diez posiciones (pixels). Para poder ejecutar esta instrucción infinitas veces recurrimos a la herramienta timer.

Además, incorporamos una función que nos permite desactivar el acelerómetro y así poder movernos con el control de dirección sin interrumpir la conducción.

```
if (e.WiimoteState.ButtonState.Left == true)
{
 wm.SetReportType(InputReport.IRAccel, false);
}
```

6.6.2. Colisiones

Como se ha he visto en el apartado de diseño 5.3.1, el renderizado háptico y la detección de colisiones son uno de los factores clave a la hora de diseñar sistemas hápticos. En nuestro caso, y dado que se trata de una aplicación desarrollada en 2D y un dispositivo háptico con un solo tipo de respuesta (vibración), sólo nos centraremos en calcular cuando el vehículo colisiona con alguno de los objetos y realizaremos un cálculo de respuesta a la colisión emitiendo mayor o menor vibración en función del tipo de colisión detectada.

En la primera versión de la aplicación pudimos detectar las colisiones porque la carretera era en línea recta y con obstáculos estáticos. El código muestra cómo detectamos dicha colisión en función de la posición del objeto y la respuesta que se enviaba.

```
int posicionX ;
    posicionX = pictureBox2.Location.X;

    if (posicionX <=304)

    {
        for (int i =0; i<20;i++){
        Control.SetRumble(true);
        Control.SetLEDs(true, true, true, true);
        }
        Control.SetRumble(false);
        Control.SetLEDs(false, false, false, false);
    }
```

En la segunda versión decidimos modificar la programación de la carretera generando curvas mediante las funciones de Bezier:

```
formGraphics.Clear(Color.Gray);
/*Linea izquierda*/
formGraphics.DrawBezier(myPen,  x1, y1-10*A, x2, y2-10*A,  x3, y3-10*A, x1, y1-5*A) ;
formGraphics.DrawLine(myPen, x1,y1-5*A,x1,y1);
formGraphics.DrawBezier(myPen,  x1, y1, x2, y2,  x3, y3, x4, y4);// la s
formGraphics.DrawBezier(myPen, xx1, yy1, xx2, yy2,  xx3, yy3, xx4, yy4);
formGraphics.DrawLine(myPen, xx4, yy4, xx4,yy4+2*A);

/*Linea discontinua*/
formGraphics.DrawBezier(myPen2, x1+A, y1-10*A, x2+A, y2-10*A,  x3+A, y3-10*A, x1+A, y1-5*A);
formGraphics.DrawLine(myPen2, x1+A,y1-5*A,x1+A,y1);
formGraphics.DrawBezier(myPen2, x1+A, y1, x2+A, y2,  x3+A, y3, x4+A, y4);
formGraphics.DrawBezier(myPen2, xx1+A, yy1, xx2+A, yy2,  xx3+A, yy3, xx4+A, yy4);
formGraphics.DrawLine(myPen2, xx4+A, yy4, xx4+A, yy4+2*A);

/*Linea derecha*/
formGraphics.DrawBezier(myPen, x1+2*A, y1-10*A, x2+2*A, y2-10*A,  x3+2*A, y3-10*A, x1+2*A, y1-5*A);
formGraphics.DrawLine(myPen, x1+2*A,y1-5*A,x1+2*A,y1);
formGraphics.DrawBezier(myPen, x1+2*A, y1, x2+2*A, y2,  x3+2*A, y3, x4+2*A, y4);
formGraphics.DrawBezier(myPen, xx1+2*A, yy1, xx2 +2*A, yy2,  xx3+2*A, yy3, xx4+2*A, yy4);
formGraphics.DrawLine(myPen, xx4+2*A, yy4, xx4+2*A, yy4+2*A);

y1++; y2++; y3++; y4++; yy1++; yy2++; yy3++; yy4++;
```

El principal problema que tuvimos fue que no podíamos localizar las colisiones entre el vehículo y la carretera. Dado que consideramos demasiado complejo implementar la solución, optamos por un tercer escenario.

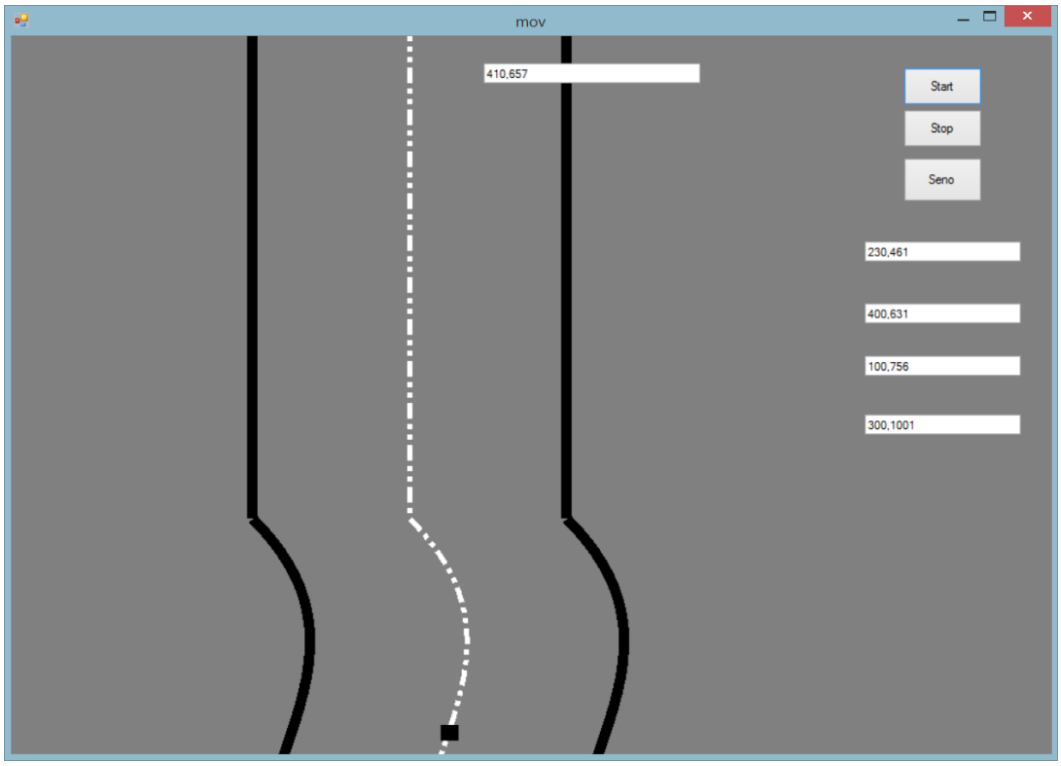


Fig. 6.7 Versión 2 Simulador de Conducción (**Fuente:** Elaboración propia)

Por ese motivo decidimos modificar el escenario e implementar la versión actual. Esta versión dispone de suficientes curvas y varios objetos colocados aleatoriamente para poder detectar colisiones.

Es importante remarcar que recurrimos a la creación de un *PictureBox* que contiene un Bitmap. A su vez, el Bitmap contiene una imagen de la carretera creada por nosotros.

```
Bitmap carretera = new Bitmap (@"G:\TFC\Imagenes mov v6_21072015\mov\mov\carretera.png");

        Graphics g = Graphics.FromImage(carretera);
        pictureBox1.Image = (Image)carretera;
```

El método *GetPixel(int x, int y)* del *Bitmap*, nos permite conocer el color exacto en una posición concreta. Nuestra solución en este caso, requiere conocer la posición del vehículo cuando éste está posicionado justo en los arcenes y la línea que diferencia ambos carriles. Para ello, utilizamos el método *PointToClient* de la clase *PictureBox* que permite conocer la posición de un objeto, en nuestro caso el vehículo, dentro de un *PictureBox*.

```
Int32 intlocalx ;
intlocalx =Convert.ToInt32( pictureBox1.PointToClient(lbl_Coche.Location).X);
Int32  intlocaly ;
intlocaly = Convert.ToInt32(pictureBox1.PointToClient(lbl_Coche.Location).Y);
```

Gracias a este método conoceremos la posición del vehículo en todo momento cuando esté sobre nuestro *PictureBox*. Cuando el vehículo esté posicionado sobre el color blanco, podremos realizar las funciones que deseemos. Las variables x e y, serán las posiciones relativas del vehículo, por lo tanto, en función del valor del color obtenido sabremos si hemos colisionado o no.

```
carretera.GetPixel(intlocalx, intlocaly).ToArgb()
```

6.6.3. Cronómetro

Al igual que en el Teclado Virtual, en la última versión de la aplicación se ha incluido un cronometro implementado con la función *timer* que mide el tiempo en segundos y minutos que nos permitirá cuantificar futuros ensayos.

6.6.4. Función Háptica

Para poder valorar como contribuye la tecnología háptica en la última versión de nuestra aplicación, se ha programado una función que permite habilitar o deshabilitar la vibración del mando.

Además para esta aplicación se han creado dos funciones que permiten diferenciar dos tipos de intensidades de vibración. Una vibración leve para los cambios de carril y otra un poco más intensa para las colisiones con los arcenes y obstáculos en la carretera.

```
void Vibrar_Suave ()
    {
                for (int i =0; i<5;i++)
                {
                wm.SetRumble(rumb);
                }
                wm.SetRumble(false);
    }
```

```
void Vibrar_Intensa ()
    {
                for (int i =0; i<20;i++)
                {
                wm.SetRumble(rumb);
                }
                wm.SetRumble(false);

    }
```

6.6.1. Exposición y análisis de los resultados

Una de las líneas de investigación más importantes en el área de la realidad virtual consiste en proveer al usuario de la capacidad natural de usar todos sus sentidos en el entorno simulado.

Como se ha comentado en el capítulo anterior, generalmente los sistemas hápticos se complementan con otras modalidades sensoriales tales como las visuales, auditivas e incluso olfativas. Por ello, para el diseño de nuestro

proyecto se ha optado por definir diferentes escenarios para ver el resultado obtenido en función del uso de distintos estímulos (vista y tacto):

- ESCENARIO 1: *Feedback* Visual
- ESCENARIO 2: *Feedback* Háptico + Visual
- ESCENARIO 3: *Feedback* Háptico
- ESCENARIO 4: Ratón y Teclado

En el primer escenario, el usuario únicamente recibe el *feedback* visual, utilizando como periférico de entrada el mando WiiMote para controlar las aplicaciones.

En el segundo escenario se añade el *feedback* háptico complementando el estímulo visual.

En el tercer escenario, se elimina el *feedback* visual, dejando sólo la respuesta táctil vibratoria. Para ello, los participantes deberán cerrar los ojos.

En el último escenario, se elimina el estímulo háptico y además realizamos el control de la aplicación del Teclado Virtual (**app1 en el experimento**) con el ratón. En el Simulador de Conducción (**app2 en el experimento**) se utiliza el control de dirección del mando WiiMote (desactivando la función del acelerómetro).

Se realizó un pequeño ejercicio con 5 usuarios de edades comprendidas entre 25 y 35 años. Para ello, cada participante tuvo que realizar las pruebas expuestas en el anexo I. Se observó que el resultado obtenido en ambas aplicaciones utilizando el escenario 2 siempre resultaba mejor. Si analizamos los resultados en función del escenario observamos que:

Escenario 1: Realizar la prueba con solo *feedback* visual resultó sencilla para los participantes en la app2 pero no tan sencilla para la app1.

Escenario 2: En la app2 no se detectaron mejoras cuantitativas ni cualitativas ya que se ha observado que dicha aplicación no previene la colisión. Sólo se envía un *feedback* háptico a modo informativo, dado que en todo momento los participantes pueden rectificar su trayectoria solo con la información visual.

En cambio sí se observaron mejoras cualitativas y cuantitativas en el uso de la app2. Cualitativamente, la mayoría de los participantes observaron que resultaba más fácil moverse por el teclado. Cuantitativamente, se observó que los participantes tardaban menos tiempo en marcar el mismo dígito.

Escenario 3: Con los ojos cerrados los participantes de la app1 lograron marcar el número empleando mayor tiempo y errores que en los otros escenarios (no se cometieron errores en los otros escenarios). No obstante, se ha observado que la respuesta háptica ha ayudado a finalizar el test. Sin embargo, en la app2 ninguno de los participantes logró finalizarlo. No obstante si intentaron modificar la trayectoria al recibir la vibración.

Escenario 4: En ambas aplicaciones fue el escenario que escogieron los participantes como más fáciles.

Los resultados más significativos de las pruebas son los siguientes (Para ver todos los resultados de los cuestionarios consultar el Anexo I):

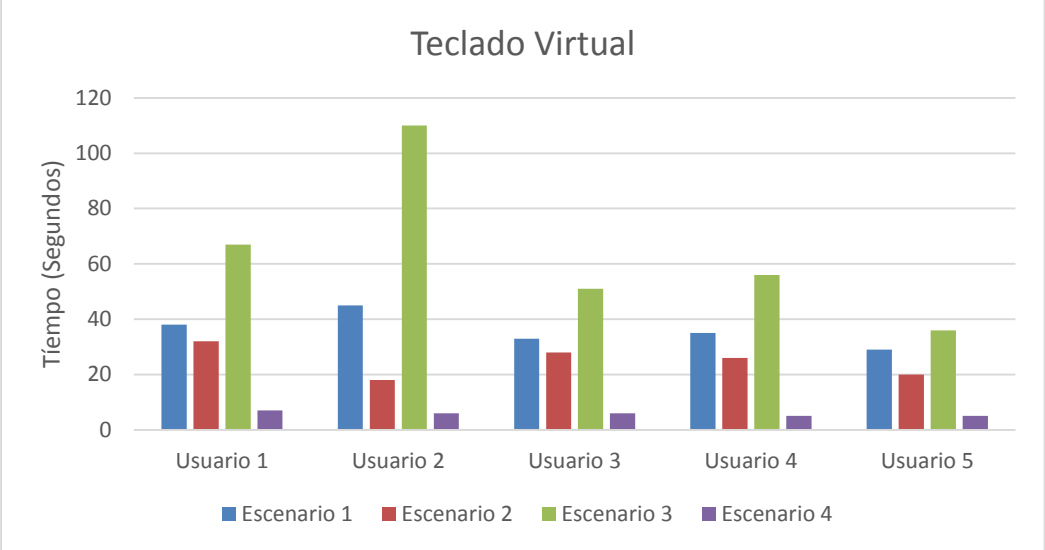


Fig. 6.8 Tiempo empleado por cada usuario en realizar la prueba (app1) por escenario (**Fuente:** Elaboración propia)

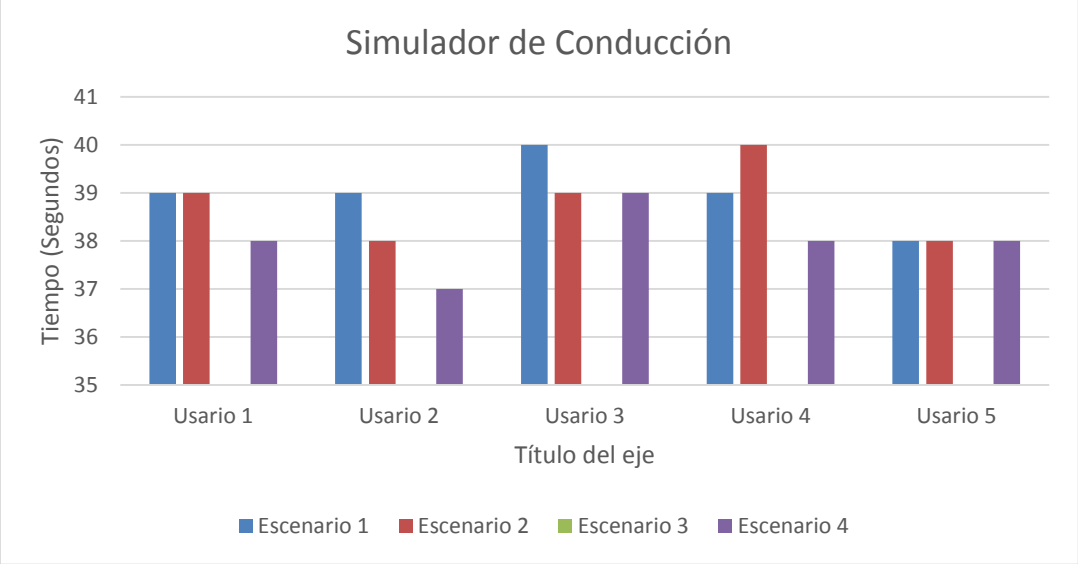


Fig. 6.9 Tiempo empleado por cada usuario en realizar la prueba (app2) por escenario (**Fuente:** Elaboración propia)

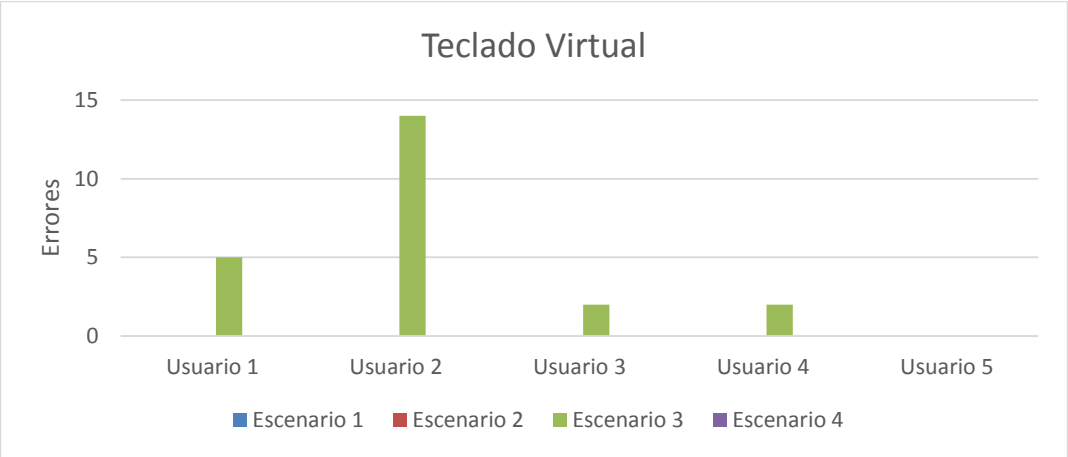


Fig. 6.10 Número de errores por cada usuario en realizar la prueba (app1) por escenario (**Fuente:** Elaboración propia)

Frente a estos resultados, nos plantemos varias cuestiones:

Todos los participantes coincidieron en que el escenario más fácil de usar era el 4. ¿Se hubiera obtenido el mismo resultado si la edad de los participantes estuviera comprendida entre los 3 y 5 años? ¿No resulta más natural desplazar un coche inclinando el mando?

Todos los participantes realizaron el ejercicio sin realizar una fase de aprendizaje previa. En la app1 se observó una clara mejora en los resultados obtenidos en el escenario 2 con respecto el escenario 1. ¿Aporta el *feedback* háptico esta mejora? Recordemos que la prueba en el escenario 2 se realizó después del escenario 1. ¿Es posible que esta fase de aprendizaje previa haya influido en los resultados? Nosotros, como usuarios expertos de nuestras aplicaciones, también obtuvimos mejores resultados en el escenario 2 que en el escenario 1.

Una de las preguntas cualitativas que realizamos en el cuestionario consistía en saber si los participantes utilizaban la vibración en los teclados de sus móviles. Todos los participantes coincidieron en la respuesta: Ninguno lo usaba. Por lo que respecta al Teclado Virtual, ¿resulta favorecedor incluir un *feedback* vibratorio cada vez que nos desplazamos por las teclas? En nuestro caso, y al tratarse de movimientos en el “aire” parece que sí. Pero, ¿cuántos de nosotros mantenemos activa la opción de respuesta por vibración al teclear en nuestros dispositivos móviles? Por norma general, un usuario habituado al uso de este tipo de dispositivos suele desactivar dicha opción. ¿El motivo? No nos aporta valor añadido ya que independientemente de acertar la tecla deseada o no, recibiremos esa respuesta.

CONCLUSIONES

7.1. Conocimientos aplicados

Durante la elaboración del Trabajo Final de Carrera, hemos podido aportar nuestros conocimientos de programación. Nunca habíamos programado tan en profundidad en lo que a nivel visual se refiere. También hemos podido aplicar nuestros conocimientos generales acerca de la VR y la AR pudiendo crear un estado del arte que nos resulta bastante interesante.

7.2. Conocimientos adquiridos

Tras haber realizado el estado del arte de VR/AR, Dispositivos de E/S y tecnología háptica, hemos profundizado más nuestros conocimientos acerca de estos tres tipos de tecnologías, pudiendo comprender hacia donde avanza la tecnología,

Durante el proyecto hemos profundizado en el lenguaje de programación C#. Hemos aprendido a utilizar elementos de Windows Forms que nunca antes habíamos visto. Controlar el foco de un elemento, desplazarlo a nuestro antojo, controlar las propiedades del bitmap.

Hemos mejorado la habilidad de trabajar en equipo, importante para la elaboración del TFC de dos personas.

7.3. Estudio de ambientalización

El Campus del Baix Llobregat ha sido diseñado con criterios medioambientales tanto en la construcción como en la utilización de los edificios. Dado que se trata de una universidad muy sensibilizada con el medio ambiente, en este apartado tendremos en cuenta el impacto ambiental que este tipo de tecnologías puede llegar a tener.

¿Será posible posible en un futuro no muy lejano generar dispositivos de entrada y salida mediante hologramas con una respuesta háptica? Sabemos que existen teclados que pueden ser proyectados en una superficie. No obstante, el hecho de no tener una respuesta táctil los hace poco prácticos. La comercialización de este tipo de dispositivos añadiendo el *feedback* háptico supondrá un ahorro en materiales y por lo tanto favorecerá el medio ambiente [69].

7.4. Futuras implementaciones

7.4.1. Mejora de la respuesta de colisión

El presente trabajo nos ha permitido experimentar como la háptica aporta un valor añadido al *software* implementado. ¿Pero es cuantificable ese valor? Para poder determinar la calidad de una tarea de accesibilidad virtual, podemos tomar como parámetro evaluador la precisión con la que se realiza. Es decir, cómo el usuario corrige el movimiento en caso de colisión. Por ello, en el caso de nuestro simulador de conducción, se podría analizar y cuantificar la penetración del vehículo en los arcenes de la carretera y así determinar mediante un experimento con diferentes usuarios como la tecnología háptica influye o no positivamente en los resultados.

7.4.2. Influencia del sonido en tareas de accesibilidad virtuales

Estudiar si la integración del sentido auditivo puede mejorar la ejecución de la tarea en el entorno virtual y analizar si los resultados obtenidos en un escenario visual + auditivo se asemejan a los de un escenario visual + háptico. Es decir, en nuestra aplicación teclado ¿obtendríamos resultados similares si en lugar de vibración se enviara un pitido cada vez que cambiamos de tecla?

7.4.3. Utilización de dispositivos hápticos

Una buena forma de completar la información aportada en este TFC sería tener la oportunidad de poder utilizar y estudiar alguno de los dispositivos hápticos comentados en la revisión del estado del arte. Dichos dispositivos disponen de librerías y *software* propios como OpenHaptics que facilitan la implementación de *software* con respuesta háptica.

7.5. Reflexión sobre la tecnología háptica

En el inicio del TFC comentamos el gran auge que tecnologías como la VR y la AR están teniendo en la actualidad [70] y el impacto a nivel tecnológico que la inversión de las grandes compañías pude tener en un futuro próximo. Durante transcurso de la realización del proyecto noticias como la compra de Metaio por parte de Apple, Canon adentrándose en el mundo de la VR o el Force Touch de los nuevos Iphone ayudan a reafirmar la pregunta que nos formulábamos durante la necesidad inicial del trabajo: ¿Es posible que nos encontremos ante un cambio tecnológico importante?

La realización del estudio del arte de los dispositivos hápticos nos ha ayudado a formarnos una idea del estado en el que se encuentra esta tecnología. En general, hemos podido observar como la mayoría de dispositivos hápticos en los que se está trabajando, únicamente tienen en cuenta su uso a través de las

manos o brazos, ya sea con un guante, con un dispositivo de escritorio o mediante texturas a través de una pantalla.

La tecnología háptica es una tecnología con un gran potencial. No obstante, ¿se podrán crear algún día dispositivos que abarquen más allá de brazo y mano? ¿Podremos sentir una inmersión lo mas realista posible? ¿Influirá el gran auge que se está viviendo de la realidad virtual y realidad aumentada en la tecnología háptica? Por ejemplo, una persona que no puede andar y su mayor ilusión es de la de alcanzar la cima del Everest, ¿no sería maravilloso que esta persona sin salir de su casa pudiera notar en sus piernas la sensación de estar escalando, sentir el frío de la nieve en sus pies y notar la textura de las rocas en su mano? Nosotros creemos que sí.

BIBLIOGRAFÍA

[1] Wikipedia, «Interacción persona-computadora,» 27 08 2015. [En línea]. Available: https://es.wikipedia.org/wiki/Interacci%C3%B3n_persona-computadora. [Último acceso: 18 09 2015].

[2] Wikipedia, «Human–computer interaction,» 12 09 2015. [En línea]. Available: https://en.wikipedia.org/w/index.php?title=Human%E2%80%93computer_interaction&oldid=680668001. [Último acceso: 18 09 2015].

[3] Wikipedia, «Head-mounted display,» 16 09 2015. [En línea]. Available: https://en.wikipedia.org/wiki/Head-mounted_display. [Último acceso: 17 09 2015].

[4] R. Metz, «Magic Leap. A startup is betting more than half a billion dollars that it will dazzle you with its approach to creating 3-D imagery,» 04 2015. [En línea]. Available: http://www.technologyreview.com/featuredstory/534971/magic-leap/. [Último acceso: 05 2015].

[5] Sony, «GDC 2015 – Novedades sobre Project Morpheus,» 03 2015. [En línea]. Available: http://blog.es.playstation.com/2015/03/04/gdc-2015-novedades-sobre-project-morpheus/. [Último acceso: 04 2015].

[6] Microsoft, «Holographic computing is here. When you change the way you see the world, you can change the world you see.,» 2015. [En línea]. Available: https://www.microsoft.com/microsoft-hololens/en-us. [Último acceso: 04 2015].

[7] M. Leap, «It's Time to Bring the Magic back into the World,» 2015. [En línea]. Available: http://www.magicleap.com/. [Último acceso: 04 2015].

[8] Oculus VR, «Rift. Next Generation Virtual Reality,» 2015. [En línea]. Available: https://www.oculus.com/en-us/. [Último acceso: 02 05 2015].

[9] Wikipedia, «Jaron Lanier,» 01 2015. [En línea]. Available: https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=Jaron_Lanier&id=679738593. [Último acceso: 03 2015].

[10] R. B. C. Manetta, Glossary of Virtual Reality Terminology, vol. 1, 1995.

[11] K. Lee, «Augmented Reality in Education and Training,» 02 2012. [En línea]. Available: http://www2.potsdam.edu/betrusak/566/Augmented%20Reality%20in%20Education.pdf. [Último acceso: 04 2015].

[12] VRS, «VRS,» 2014. [En línea]. Available: http://www.vrs.org.uk/. [Último acceso: 23 04 2015].

[13] C. Hartman, «Eric M. Howlett, pioneer of “Virtual Reality”,» 27 01 2012. [En línea]. Available: http://hightechhistory.com/tag/cyberface/. [Último acceso: 17 09 2015].

[14] Wikipedia, «Cave Automatic Virtual Environment,» 5 09 2014. [En línea]. Available: https://es.wikipedia.org/w/index.php?title=Especial:Citar&page=Cave_Automatic_Virtual_Environment&id=76796628. [Último acceso: 04 2015].

[15] Wikipedia, «Virtual Boy,» 26 09 2014. [En línea]. Available: https://es.wikipedia.org/w/index.php?title=Virtual_Boy&oldid=77205380. [Último acceso: 25 04 2015].

[16] Navy, «Augmented Reality,» 2014. [En línea]. Available: http://www.nrl.navy.mil/itd/imda/research/5581/augmented-reality. [Último acceso: 08 2015].

[17] C. L. a. O. B. M. Möhring, «Video See-Through AR on Consumer Cell-Phones,» 2004. [En línea]. Available: http://140.78.90.140/medien/ar/Pub/Cell_Phone_AR.pdf. [Último acceso: 16 04 2015].

[18] Wikipedia, «Wikitude,» 23 06 2015. [En línea]. Available: https://en.wikipedia.org/wiki/Wikitude. [Último acceso: 17 09 2015].

[19] E. Press, «Google Glass sigue preocupando por sus implicaciones en la privacidad,» 03 07 2013. [En línea]. Available: http://www.abc.es/tecnologia/informatica-hardware/20130703/abci-google-glass-dudas-privacidad-201307031042.html. [Último acceso: 09 2015].

[20] J. Carden, «History of Virtual Reality and Augmented Reality Timeline,» 04 2015. [En línea].

Available: http://www.zenka.org/history-of-virtual-reality-and-augmented-reality-timeline/. [Último acceso: 17 09 2015].

[21] Kickstarter, «Oculus Rift,» 08 2012. [En línea]. Available: https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game?ref=nav_search. [Último acceso: 12 04 2015].

[22] E. P. d. Catalunya, «Facebook compra la empresa de realidad virtual Oculus VR,» 26 03 2014. [En línea]. Available: http://www.elperiodico.com/es/noticias/economia/facebook-compra-empresa-realidad-virtual-oculusvr-3219469. [Último acceso: 04 2015].

[23] CNN, «Magic Leap releases stunning holographic video,» 20 03 2015. [En línea]. Available: http://money.cnn.com/2015/03/20/technology/magic-leap/. [Último acceso: 17 09 2015].

[24] E. A. L. Ivan, «FOVE, cuando la realidad virtual obedece a tus ojos,» 05 2015. [En línea]. Available: http://www.elandroidelibre.com/2015/05/fove-cuando-la-realidad-virtual-obedece-a-tus-ojos.html?utm_content=buffer91109&utm_medium=social&utm_source=plus.google.com&utm_campaign=buffer. [Último acceso: 05 2015].

[25] Dave Altavilla, «Apple Further Legitimizes Augmented Reality Tech With Acquistion Of Metaio,» 30 05 2015. [En línea]. Available: http://www.forbes.com/sites/davealtavilla/2015/05/30/apple-further-legitimizes-augmented-reality-tech-with-acquistion-of-metaio/. [Último acceso: 30 05 2015].

[26] Kzero, «Kzero,» 2014. [En línea]. Available: http://www.kzero.co.uk/blog/consumer-virtual-reality-market-worth-13bn-2018/. [Último acceso: 26 07 2015].

[27] R. Castañeda, «Software y hardware,» [En línea]. Available: http://softhardaa.blogspot.com.es/p/hardware-perifericos-de-entrada-y-de.html . [Último acceso: 10 05 2015].

[28] L. Zevallos, «Monografía de la Impresora,» [En línea]. Available: http://www.monografias.com/trabajos59/impresora/impresora.shtml. [Último acceso: 18 09 2015].

[29] B. Riken, «IN/OUT Interface,» [En línea]. Available: http://www.brain.riken.jp/bsi-news/bsinews34/no34/research1e.html. [Último acceso: 08 2015].

[30] M. BENÍTEZ, «FeelReal: Olores para la realidad virtual,» 12 03 2015. [En línea]. Available: http://www.neoteo.com/feelreal-olores-para-la-realidad-virtual/. [Último acceso: 04 2015].

[31] Tekever , «Tekever,» 2015. [En línea]. Available: www.tekever.com. [Último acceso: 17 08 2015].

[32] BrianFlight, «BrianFlight,» 2015. [En línea]. Available: www.brianflight.org. [Último acceso: 17 08 2015].

[33] Ecured, «Tecnología háptica,» [En línea]. Available: http://www.ecured.cu/index.php/Tecnolog%C3%ADa_h%C3%A1ptica. [Último acceso: 30 04 2015].

[34] J. Martínez, J. P. Molina, . A. S. García, D. Martínez y P. Gonzalez, «Desarrollo de un Guante de Datos con Retorno Háptico,» [En línea]. Available: http://jonatanmartinez.com/papers/Martinez2009.pdf. [Último acceso: 29 04 2015].

[35] R. A. Española, «rae,» [En línea]. Available: http://www.rae.es/. [Último acceso: 25 Julio 2015].

[36] S. BALLESTEROS, «PERCEPCIÓN HAPTICA DE OBJETOS Y PATRONES REALZADOS: UNA REVISIÓN,» *Psicothema,* vol. 5, nº 2, pp. 311-321, 1993.

[37] Wikipedia, «Tacto,» 13 Julio 2015. [En línea]. Available: https://es.wikipedia.org/w/index.php?title=Tacto&oldid=83768675. [Último acceso: 29 Julio 2015].

[38] k. Memi, «Sinestesia, Kinestesia Y Cenestesia,» ClubEnsayos.com, Julio 2015. [En línea]. Available: https://www.clubensayos.com/Psicología/Sinestesia-Kinestesia-Y-Cenestesia/2629021.html. [Último acceso: 25 Julio 2015].

[39] M. Garat, «Biología. Receptores de la piel,» 15 06 2014. [En línea]. Available: http://biologiaprofesoramaite.blogspot.com.es/. [Último acceso: 25 Mayo 2015].

[40] T. Dezcallar, «Relación entre procesos mentales y sentido háptico,» 03 2012. [En línea].

Available: http://www.tdx.cat/bitstream/handle/10803/96819/tds1de1.pdf?sequence=1. [Último acceso: 12 05 2015].

[41] D. L. F. Pacheco, «Universidad de Costa Rica. Facultad de Medicina,» 2007. [En línea]. Available: http://163.178.103.176/Fisiologia/neuro_prac_bas_p7_11.html. [Último acceso: 25 Mayo 2015].

[42] K. B. Shimoga, «Finger Force and Touch Feedback Issues in Dexterous Telemanipulation,» de *Fourth Annual Conference on Intelligent Robotic Systems for Space Exploration*, 1992.

[43] Microsoft, «Microsoft Research,» [En línea]. Available: http://research.microsoft.com/en-us/news/features/haptics-080514.aspx. [Último acceso: 03 2015].

[44] P. Ciáurriz, «HAPTICALLY-COUPLED DEVICES: STABILITY ANALYSIS AND APPLICATION TO DRIVE-BY-WIRE SYSTEMS,» 12 12 2014. [En línea]. Available: http://dadun.unav.edu/handle/10171/37205.

[45] Valve, «Steam Controller,» 2015. [En línea]. Available: http://store.steampowered.com/universe/controller/. [Último acceso: 08 2015].

[46] K. Steinberg, «Cambridge Research & Development,» 02 2013. [En línea]. Available: http://www.cambridgerad.com/CRD-BCH-Haptic-Release-2013.pdf. [Último acceso: 26 04 2015].

[47] Intuitive Surgical, «Intuitive Surgical,» 2015. [En línea]. Available: http://www.intuitivesurgical.com/. [Último acceso: 26 04 2015].

[48] davincisurgery, «da vinci Surgery,» 2015. [En línea]. Available: http://www.davincisurgery.com/. [Último acceso: 26 04 2015].

[49] CyberGlove, «CyberGlove,» 2015. [En línea]. Available: http://www.cyberglovesystems.com/. [Último acceso: 26 05 2015].

[50] Geomagic, «Geomagic,» [En línea]. Available: www.geomagic.com. [Último acceso: 04 2015].

[51] J. J. Gibson, Observations on active touch. Psycological review, vol. 69, 1962, pp. 477-490.

[52] E. Huelgas, «Tecnología Háptica,» 04 2013. [En línea]. Available: https://htid3.files.wordpress.com/2013/04/tecnologia-haptica.pdf. [Último acceso: 27 04 2015].

[53] Neuro Digital, «Neuro Digital,» [En línea]. Available: www.neurodigital.es. [Último acceso: 06 2015].

[54] Neuro Digital Technologies, «GloveOne,» 2015. [En línea]. Available: www.gloveonevr.com. [Último acceso: 04 2015].

[55] OSVR, «OSVR,» [En línea]. Available: www.osvr.org. [Último acceso: 01 08 2015].

[56] Samsung, «Samsung,» [En línea]. Available: http://www.samsung.com/es/consumer/mobile-devices/wearables/gear/SM-R320NPWAPHE. [Último acceso: 08 2015].

[57] Apple, «Iphone 6S,» 09 2015. [En línea]. Available: http://www.apple.com/es/iphone-6s/3d-touch/. [Último acceso: 09 2015].

[58] Puskarich; Paul G., «US Patent & Trademark Office,» 23 04 2015. [En línea]. Available: http://appft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=HITOFF&p=1&u=%2Fnetahtml%2FPTO%2Fsearch-adv.html&r=17&f=G&l=50&d=PG01&S1=(345%2F173.CCLS.+AND+20150423.PD.)&OS=ccl/345/173+and+pd/4/23/2015&RS=(CCL/345/173+AND+PD/20150423). [Último acceso: 12 09 2015].

[59] Disney, «Aireal,» [En línea]. Available: http://www.disneyresearch.com/project/aireal/. [Último acceso: 29 08 2015].

[60] J. Hernantes, «INTERACCIÓN EN SISTEMAS HÁPTICOS MULTISENSORIALES: RESPUESTA DE COLISIÓN Y MEJORAS DE USABILIDAD,» 10 2011. [En línea]. Available: http://dadun.unav.edu/handle/10171/19153. [Último acceso: 08 2015].

[61] I. Diaz, «Métodos de renderizado multisensorial y análisis de estabilidad en interfaces hápticos,» 2008. [En línea]. Available: http://dadun.unav.edu/handle/10171/15650. [Último acceso: 08 2015].

[62] Wikipedia, «Seis Grados de Libertad,» 3 04 2014. [En línea]. Available: https://es.wikipedia.org/wiki/Seis_grados_de_libertad. [Último acceso: 10 09 2015].

[63] «arkigrafico,» [En línea]. Available: http://www.arkigrafico.com/definicion-de-render-que-es-renderizacion/#. [Último acceso: 1 9 2015].

[64] Cornell University, «Haptic Glove,» [En línea]. Available: https://courses.cit.cornell.edu/ee476/FinalProjects/s2008/crs54_tz36/crs54_tz36/twocolumn.html. [Último acceso: 03 2015].

[65] Brian Peek, «Managed Library for Nintendo's Wiimote,» 12 2013. [En línea]. Available: https://wiimotelib.codeplex.com/. [Último acceso: 05 2015].

[66] SharpDevelop, «http://www.icsharpcode.net/OpenSource/SD/Default.aspx,» 01 06 2015. [En línea]. Available: http://www.icsharpcode.net/OpenSource/SD/Default.aspx. [Último acceso: 01 06 2015].

[67] J. I. M. Martinez, «blog.jorgeivanmeza.com,» 12 2008. [En línea]. Available: http://blog.jorgeivanmeza.com/2008/12/utilizando-el-wiimote-con-csharp/. [Último acceso: 06 2015].

[68] J. Skeet, «Threading in Windows Forms,» [En línea]. Available: http://www.yoda.arachsys.com/csharp/threads/winforms.shtml. [Último acceso: 06 2015].

[69] W. Wolz, «Up next: A Wearable Holodeck ... with actual feels!,» 05 2015. [En línea]. Available: http://www.google.com/url?q=http%3A%2F%2Fbbs.boingboing.net%2Ft%2Fup-next-a-wearable-holodeck-with-actual-feels%2F31720&sa=D&sntz=1&usg=AFQjCNEONSeFSmcyt3iA6sGrfwME3cp1pA. [Último acceso: 09 2015].

[70] K. Puerto, «Canon también se apunta a la realidad virtual, pero pasa de cascos,» 13 09 2015. [En línea]. Available: http://www.xataka.com/realidad-virtual-aumentada/canon-tambien-se-apunta-a-la-realidad-virtual-pero-pasa-de-cascos. [Último acceso: 13 09 2015].

ANEXOS

ANEXO I. Cuestionarios

1- Plantilla:

Dentro del experimento se plantean tres escenarios comentados en el capítulo de implementación:

ESCENARIO 1: *Feedback* Visual
ESCENARIO 2: *Feedback* Háptico + Visual
ESCENARIO 3: *Feedback* Háptico
ESCENARIO 4: Ratón y Teclado

Cuestionario previo:

- Edad:
- ¿Has utilizado alguna vez la Wii? (si/no)
- ¿Conoces el término háptico? (si/no)
- ¿Utilizas la vibración en el teclado de tu móvil? (si/no)

Aplicación Teclado Virtual:

Preguntas cualitativas. Valora del 1 al 5 (siendo 1 muy en desacuerdo y 5 muy de acuerdo)

- ¿Te ha resultado difícil moverte por el teclado?

  Escenario 1: 1 2 3 4 5
  Escenario 2: 1 2 3 4 5
  Escenario 3: 1 2 3 4 5
  Escenario 4: 1 2 3 4 5

- Ordena los escenarios por facilidad de uso (Siendo 1 el más fácil y 4 el más complejo):

  Escenario 1:
  Escenario 2:
  Escenario 3:
  Escenario 4:

Preguntas cuantitativas.

- Tiempo que tarda el usuario en marcar el número 695

  Escenario 1:
  Escenario 2:
  Escenario 3:
  Escenario 4:

- Numero de errores al marcar el número 695

  Escenario 1:
  Escenario 2:
  Escenario 3:
  Escenario 4:

Aplicación Simulador de Conducción:

- ¿Te ha resultado difícil moverte?

  Escenario 1: 1 2 3 4 5
  Escenario 2: 1 2 3 4 5
  Escenario 3: 1 2 3 4 5
  Escenario 4: 1 2 3 4 5

- Ordena los escenarios por facilidad de uso (Siendo 1 el más fácil y 4 el más complejo):

  Escenario 1:
  Escenario 2:
  Escenario 3:
  Escenario 4:

Preguntas cuantitativas.

- Número veces que el usuario se sale del arcén en cada escenario.

  Escenario 1:
  Escenario 2:
  Escenario 3:
  Escenario 4:

- Tiempo que tarda el usuario en finalizar la carrera

  Escenario 1:
  Escenario 2:
  Escenario 3:
  Escenario 4:

Preguntas Abiertas.

- ¿Qué dos o tres cosas han gustado más del experimento?
- ¿Qué dos o tres cosas te han gustado menos del experimento?

2- Cuestionarios rellenados por los participantes del test. (usuario 1)

Cuestionario previo:

- Edad: 26
- ¿Has utilizado alguna vez la Wii? **Si**
- ¿Conoces el término háptico? **Si**
- ¿Utilizas la vibración en el teclado de tu móvil? **No**

Aplicación Teclado Virtual:

Preguntas cualitativas. Valora del 1 al 5 (siendo 1 muy en desacuerdo y 5 muy de acuerdo)

- ¿Te ha resultado difícil moverte por el teclado?

  Escenario 1: 1 2 ~~3~~ 4 5
  Escenario 2: 1 ~~2~~ 3 4 5
  Escenario 3: 1 2 3 4 ~~5~~
  Escenario 4: ~~1~~ 2 3 4 5

- Ordena los escenarios por facilidad de uso (Siendo 1 el más fácil y 4 el más complejo):

  Escenario 1: 3
  Escenario 2: 2
  Escenario 3: 4
  Escenario 4: 1

Preguntas cuantitativas.

- Tiempo que tarda el usuario en marcar el número 695

  Escenario 1: 38
  Escenario 2: 32
  Escenario 3: 67
  Escenario 4: 7

- Numero de errores al marcar el número 695

  Escenario 1: 0
  Escenario 2: 0
  Escenario 3: 5
  Escenario 4: 0

Aplicación Simulador de Conducción:

¿Te ha resultado difícil moverte?

Escenario 1: 1 2 ~~3~~ 4 5

Escenario 2: 1 ~~2~~ 3 4 5
Escenario 3: 1 2 3 4 ~~5~~
Escenario 4: ~~1~~ 2 3 4 5

- Ordena los escenarios por facilidad de uso (Siendo 1 el más fácil y 4 el más complejo):

  Escenario 1: 3
  Escenario 2: 2
  Escenario 3: 4
  Escenario 4: 1

Preguntas cuantitativas.

- Número veces que el usuario se sale del arcén en cada escenario.

  Escenario 1: 0
  Escenario 2: 0
  Escenario 3: 1
  Escenario 4: 0

- Tiempo que tarda el usuario en finalizar la carrera

  Escenario 1: 39
  Escenario 2: 39
  Escenario 3: --
  Escenario 4: 38

Preguntas Abiertas.

- ¿Qué dos o tres cosas han gustado más del experimento?
  - Competitivo, se notaba la vibración
- ¿Qué dos o tres cosas te han gustado menos del experimento?
  - Obstáculos móviles

3- Cuestionarios rellenados por los participantes del test. (usuario 2)

Cuestionario previo:

- Edad: 32
- ¿Has utilizado alguna vez la Wii? **Si**
- ¿Conoces el término háptico? **Si**
- ¿Utilizas la vibración en el teclado de tu móvil? **No**

Aplicación Teclado Virtual:

Preguntas cualitativas. Valora del 1 al 5 (siendo 1 muy en desacuerdo y 5 muy de acuerdo)

- ¿Te ha resultado difícil moverte por el teclado?

  Escenario 1: 1 2 ~~3~~ 4 5
  Escenario 2: 1 ~~2~~ 3 4 5
  Escenario 3: 1 2 3 4 ~~5~~
  Escenario 4: ~~1~~ 2 3 4 5

- Ordena los escenarios por facilidad de uso (Siendo 1 el más fácil y 4 el más complejo):

  Escenario 1: 2
  Escenario 2: 3
  Escenario 3: 4
  Escenario 4: 1

Preguntas cuantitativas.

- Tiempo que tarda el usuario en marcar el número 695

  Escenario 1: 45
  Escenario 2: 18
  Escenario 3: 110
  Escenario 4: 6

- Numero de errores al marcar el número 695

  Escenario 1: 0
  Escenario 2: 0
  Escenario 3: 14
  Escenario 4: 0

Aplicación Simulador de Conducción:

- ¿Te ha resultado difícil moverte?

  Escenario 1: 1 2 ~~3~~ 4 5

Escenario 2: 1 ~~2~~ 3 4 5
Escenario 3: 1 2 3 4 ~~5~~
Escenario 4: ~~1~~ 2 3 4 5

- Ordena los escenarios por facilidad de uso (Siendo 1 el más fácil y 4 el más complejo):

  Escenario 1: 3
  Escenario 2: 2
  Escenario 3: 4
  Escenario 4: 1

Preguntas cuantitativas.

- Número veces que el usuario se sale del arcén en cada escenario.

  Escenario 1: 0
  Escenario 2: 0
  Escenario 3: 1
  Escenario 4: 0

- Tiempo que tarda el usuario en finalizar la carrera

  Escenario 1: 39
  Escenario 2: 38
  Escenario 3: --
  Escenario 4: 37

Preguntas Abiertas.

- ¿Qué dos o tres cosas han gustado más del experimento?
  - Divertido, interesante

- ¿Qué dos o tres cosas te han gustado menos del experimento?
  - Difícil, había que girar mucho el mando para poder girar el coche y el teclado.

4- Cuestionarios rellenados por los participantes del test. (usuario 3)

Cuestionario previo:

- Edad: 34
- ¿Has utilizado alguna vez la Wii? **Si**
- ¿Conoces el término háptico? **No**
- ¿Utilizas la vibración en el teclado de tu móvil? **No**

Aplicación Teclado Virtual:

Preguntas cualitativas. Valora del 1 al 5 (siendo 1 muy en desacuerdo y 5 muy de acuerdo)

- ¿Te ha resultado difícil moverte por el teclado?

  Escenario 1: 1 2 ~~3~~ 4 5
  Escenario 2: 1 ~~2~~ 3 4 5
  Escenario 3: 1 2 3 4 ~~5~~
  Escenario 4: ~~1~~ 2 3 4 5

- Ordena los escenarios por facilidad de uso (Siendo 1 el más fácil y 4 el más complejo):

  Escenario 1: 3
  Escenario 2: 2
  Escenario 3: 4
  Escenario 4: 1

Preguntas cuantitativas.

- Tiempo que tarda el usuario en marcar el número 695

  Escenario 1: 33
  Escenario 2: 28
  Escenario 3: 51
  Escenario 4: 6

- Numero de errores al marcar el número 695

  Escenario 1: 0
  Escenario 2: 0
  Escenario 3: 2
  Escenario 4: 0

Aplicación Simulador de Conducción:

- ¿Te ha resultado difícil moverte?

Escenario 1: 1 ~~2~~ 3 4 5
Escenario 2: 1 2 ~~3~~ 4 5
Escenario 3: 1 2 3 4 ~~5~~
Escenario 4: ~~1~~ 2 3 4 5

- Ordena los escenarios por facilidad de uso (Siendo 1 el más fácil y 4 el más complejo):

Escenario 1: 2
Escenario 2: 3
Escenario 3: 4
Escenario 4: 1

Preguntas cuantitativas.

- Número veces que el usuario se sale del arcén en cada escenario.

Escenario 1: 0
Escenario 2: 0
Escenario 3: 1
Escenario 4: 0

- Tiempo que tarda el usuario en finalizar la carrera

Escenario 1: 40
Escenario 2: 39
Escenario 3: --
Escenario 4: 39

Preguntas Abiertas.

- ¿Qué dos o tres cosas han gustado más del experimento?
  - Divertido
- ¿Qué dos o tres cosas te han gustado menos del experimento?

5- Cuestionarios rellenados por los participantes del test. (usuario 4)

Cuestionario previo:

- Edad: 30
- ¿Has utilizado alguna vez la Wii? **Si**
- ¿Conoces el término háptico? **Si**
- ¿Utilizas la vibración en el teclado de tu móvil? **No**

Aplicación Teclado Virtual:

Preguntas cualitativas. Valora del 1 al 5 (siendo 1 muy en desacuerdo y 5 muy de acuerdo)

- ¿Te ha resultado difícil moverte por el teclado?

  Escenario 1: 1 2 ~~3~~ 4 5
  Escenario 2: 1 ~~2~~ 3 4 5
  Escenario 3: 1 2 3 4 ~~5~~
  Escenario 4: ~~1~~ 2 3 4 5

- Ordena los escenarios por facilidad de uso (Siendo 1 el más fácil y 4 el más complejo):

  Escenario 1: 3
  Escenario 2: 2
  Escenario 3: 4
  Escenario 4: 1

Preguntas cuantitativas.

- Tiempo que tarda el usuario en marcar el número 695

  Escenario 1: 35
  Escenario 2: 26
  Escenario 3: 56
  Escenario 4: 5

- Numero de errores al marcar el número 695

  Escenario 1: 0
  Escenario 2: 0
  Escenario 3: 2
  Escenario 4: 0

Aplicación Simulador de Conducción:

- ¿Te ha resultado difícil moverte?

Escenario 1: 1 ~~2~~ 3 4 5
Escenario 2: 1 2 ~~3~~ 4 5
Escenario 3: 1 2 3 4 ~~5~~
Escenario 4: ~~1~~ 2 3 4 5

- Ordena los escenarios por facilidad de uso (Siendo 1 el más fácil y 4 el más complejo):

Escenario 1: 2
Escenario 2: 3
Escenario 3: 4
Escenario 4: 1

Preguntas cuantitativas.

- Número veces que el usuario se sale del arcén en cada escenario.

Escenario 1: 0
Escenario 2: 0
Escenario 3: 1
Escenario 4: 0

- Tiempo que tarda el usuario en finalizar la carrera

Escenario 1: 39
Escenario 2: 40
Escenario 3: --
Escenario 4: 38

Preguntas Abiertas.

- ¿Qué dos o tres cosas han gustado más del experimento?
  - El teclado es competitivo
- ¿Qué dos o tres cosas te han gustado menos del experimento?
  - Una carrera con más coches.

6- Cuestionarios rellenados por los participantes del test. (usuario 5)

Cuestionario previo:

- Edad: 28
- ¿Has utilizado alguna vez la Wii? **Si**
- ¿Conoces el término háptico? **No**
- ¿Utilizas la vibración en el teclado de tu móvil? **No**

Aplicación Teclado Virtual:

Preguntas cualitativas. Valora del 1 al 5 (siendo 1 muy en desacuerdo y 5 muy de acuerdo)

- ¿Te ha resultado difícil moverte por el teclado?

  Escenario 1: 1 2 ~~3~~ 4 5
  Escenario 2: 1 ~~2~~ 3 4 5
  Escenario 3: 1 2 3 4 ~~5~~
  Escenario 4: ~~1~~ 2 3 4 5

- Ordena los escenarios por facilidad de uso (Siendo 1 el más fácil y 4 el más complejo):

  Escenario 1: 3
  Escenario 2: 2
  Escenario 3: 4
  Escenario 4: 1

Preguntas cuantitativas.

- Tiempo que tarda el usuario en marcar el número 695

  Escenario 1: 29
  Escenario 2: 20
  Escenario 3: --
  Escenario 4: 5

- Numero de errores al marcar el número 695

  Escenario 1: 0
  Escenario 2: 0
  Escenario 3: 0
  Escenario 4: 0

Aplicación Simulador de Conducción:

- ¿Te ha resultado difícil moverte?

  Escenario 1: 1 2 ~~3~~ 4 5

Escenario 2: 1 ~~2~~ 3 4 5
Escenario 3: 1 2 3 4 ~~5~~
Escenario 4: ~~1~~ 2 3 4 5

- Ordena los escenarios por facilidad de uso (Siendo 1 el más fácil y 4 el más complejo):

Escenario 1: 3
Escenario 2: 2
Escenario 3: 4
Escenario 4: 1

Preguntas cuantitativas.

- Número veces que el usuario se sale del arcén en cada escenario.

Escenario 1: 0
Escenario 2: 0
Escenario 3: 1
Escenario 4: 0

- Tiempo que tarda el usuario en finalizar la carrera

Escenario 1: 38
Escenario 2: 38
Escenario 3: 38
Escenario 4: 38

Preguntas Abiertas.

- ¿Qué dos o tres cosas han gustado más del experimento?
  - Fácil
- ¿Qué dos o tres cosas te han gustado menos del experimento?
  - Fácil, falta de más coches en la carretera

ANEXO II. Conexión de WiiMote con el equipo

1- Requisitos de conexión:

Para poder utilizar las aplicaciones será necesario disponer de un ordenador con:

Windows 8.1
Conexión Bluetooth
Opcionalmente el *software* SharpDevelop

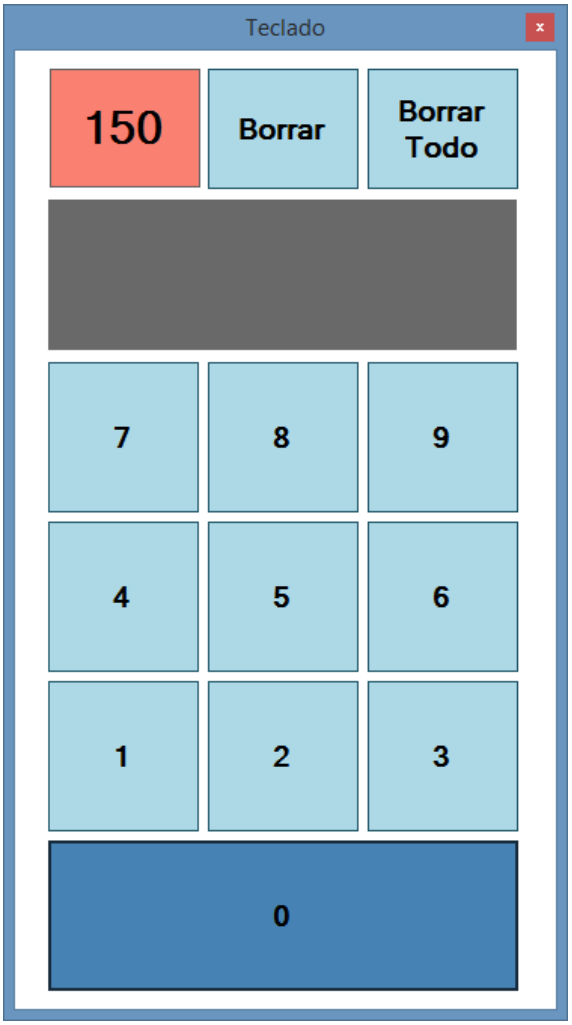
2- Conectar el mando con el PC

En primer lugar, será necesario establecer la conexión vía bluetooth del mando Wiimote y el ordenador. Para ello, se deberá ir a “Administrar dispositivos Bluetooth” de Windows, y mientras se presionan las teclas 1 y 2 del mando vincular los dispositivos.

ANEXO III. Manual de usuario del Teclado Virtual

1- **Ejecución de la aplicación**.

Se deberá ejecutar el fichero Teclado150611.exe ubicado dentro de la carpeta del proyecto \bin\Debug.



2- Uso de la aplicación.

Para el uso correcto del teclado, será necesario colocar el mando horizontalmente orientado a la izquierda. Tal y como se muestra en la imagen:

Para desplazamientos sobre el teclado hacia la derecha, deberemos inclinar el dispositivo 90 grados en esa dirección. Y al revés para desplazamientos en dirección izquierda.

Una vez situados sobre la tecla deseada, se deberá presionar la tecla A del mando para poder pulsar la tecla virtual. Una vez hecho, aparecerá el dígito por pantalla, se reproducirá la transcripción del número y se recibirá una pequeña vibración.

Para borrar el último dígito introducido deberá presionarse el botón “Borrar”. Si por lo contrario se desea borrar todos los dígitos que aparecen en pantalla y reiniciar el cronometro, deberá presionarse la tecla “Borrar Todo”.

3- Deshabilitar Háptica

Para poder probar la aplicación sin tecnología háptica, se deberá pulsar la tecla “-“del mando. Para volver a activarla deberá presionarse la tecla “+”. Por defecto, dicha función viene activa.

ANEXO IV. Manual de usuario del Simulador de Conducción

1- Ejecución de la aplicación.

Se deberá ejecutar el fichero Mov.exe ubicado dentro de la carpeta del proyecto \bin\Debug.

2- Uso de la aplicación

Para iniciar una carrera se deberá pulsar el botón “1” del mando. Para finalizarla inmediatamente se deberá pulsar el botón “2”.

Para el uso correcto de la aplicación, será necesario colocar el mando horizontalmente orientado a la izquierda. Tal y como se muestra en la imagen:

Para desplazamientos hacia la derecha, deberemos inclinar el dispositivo más de 45 grados en esa dirección. Y al revés para desplazamientos en dirección contraria.

Durante el desplazamiento del vehículo, se recibirán diferentes intensidades de vibración dependiendo si existe colisión con algún objeto o cuando el vehículo se sitúa fuera carretera.

Para deshabilitar el acelerómetro y controlar el vehículo con el control de dirección se deberá pulsar tecla de dicho control dirección “arriba”. Para volver a habilitarlo, se deberá pulsar la tecla “Abajo”

El juego finaliza cuando se llega a la meta o cuando nos salimos de la carretera.

3- Deshabilitar Háptica

Para poder probar la aplicación sin tecnología háptica, se deberá pulsar la tecla “-“del mando. Para volver a activarla deberá presionarse la tecla “+”. Por defecto, dicha función viene activa.

ANEXO V. Datasheet Geomagic

Tabla especificaciones dispositivos Hápticos Geomagic:

Geomagic Haptic Device Specifications

Geomagic Touch y Geomagic Touch X	**Geomagic Touch**	**Geomagic Touch X**
		
Área de trabajo	~ 6,4 An x 4,8 Al x 2,8 P pulg. > 160 An x 120 Al x 70 P mm	~ 6,4 An x 4,8 Al x 4,8 P pulg. > 160 An x 120 Al x 120 P mm
Rango de movimiento	Movimiento de la mano con giro de la muñeca	Movimiento de la mano con giro de la muñeca
Resolución de la posición nominal	> 450 dpi ~ 0,055 mm	> 1100 dpi ~ 0,023 mm
Fuerza de esfuerzo máxima en la posición nominal (brazos ortogonales)	0,75 lbf. (3,3 N)	1,8 lbf. (7,9 N)
Rigidez	Eje X > 7,3 lb/pulg. (1,26 N/mm) Eje Y > 13,4 lb/pulg. (2,31 N/mm) Eje Z > 5,9 lb/pulg. (1,02 N/mm)	Eje X > 10,8 lb/pulg. (1,86 N/mm) Eje Y > 13,6 lb/pulg. (2,35 N/mm) Eje Z > 8,6 lb/pulg. (1,48 N/mm)
Retroalimentación de fuerza (3 grados de libertad)	x, y, z	x, y, z
Entrada/sensor de posición (6 grados de libertad) [Gimbal del lápiz]	x, y, z (codificadores digitales) [Inclinación, giro, dirección (± 5% de potenciómetros de linealidad)]	x, y, z (codificadores digitales) [Inclinación, giro, dirección (± 3% de potenciómetros de linealidad)]
Interfaz	Puerto IEEE-1394 FireWire®: 6 pines a 6 pines	Puerto IEEE-802.3 Ethernet

Phantom Premiums	**Premium 1.0**	**Premium 1.5**	**1.5 High Force (HF)**	**Premium 3.0**
				
Área de trabajo	10 An x 7 Al x 5 P pulgadas 254 An x 178 Al x 127 P mm	15 An x 10,5 Al x 7,5 P pulgadas 381 An x 267 Al x 191 P mm		33 An x 23 Al x 16 P pulgadas 838 An x 584 Al x 406 P mm
Rango de movimiento	Movimiento de la mano con giro de la muñeca	Movimiento de brazo inferior con giro del codo		Movimiento de brazo completo con giro del hombro
Resolución de la posición nominal	860 dpi 0,03 mm	860 dpi 0,03 mm	3784 dpi 0,007 mm	> 1000 dpi ~ 0,02 mm
Fuerza de esfuerzo máxima (posición nominal)	1,9 lbf 8,5 N	1,9 lbf 8,5 N	8,4 lbf 37,5 N	4,9 lbf 22 N
Rigidez	20 lbf pulg.$^{-1}$ 3,5 N mm^{-1}	20 lbf pulg.$^{-1}$ 3,5 N mm^{-1}		5,7 lbf pulg.$^{-1}$ 1 N mm^{-1}
Retroalimentación de fuerza (3 grados de libertad)	x, y, z	x, y, z		x, y, z
Entrada/sensor de posición (3 grados de libertad con 3 grados adicionales opcionales)	x, y, z (inclinación, giro y dirección opcional)	x, y, z (inclinación, giro y dirección opcional)	x, y, z (inclinación, giro y dirección opcional)	x, y, z (inclinación, giro y dirección opcional)
Interfaz	Puerto paralelo	Puerto paralelo	Puerto paralelo	Puerto paralelo
End effectors opcionales	Encoder Gimbal	Encoder Gimbal	Encoder Gimbal HF	Encoder Gimbal

ANEXO VI. Datasheet CyberGlove

1- Cybertouch

CyberGlove Systems

CyberTouch™
Tactile Feedback for the CyberGlove System

Add vibro-tactile feedback to the CyberGlove® system, the industry-leading electronic data glove.

The CyberTouch™ feedback option enables CyberGlove® users to manually experience virtual worlds, feeling vibro-tactile sensations from interaction with computer generated 3D objects. The addition of vibro-tactile feedback creates a more realistic environment for users, enabling them to experience how a virtual object moves and responds to interaction. The CyberTouch system can also be used for data visualization to feel vibrational intensity proportional to ground-density data, water content, magnetic field strength, hazard proximity, or even light intensity.

Specifications

Vibro-tactile actuators: 6; one on each finger, one on the palm

Vibrational Frequency: 0 – 125 Hz

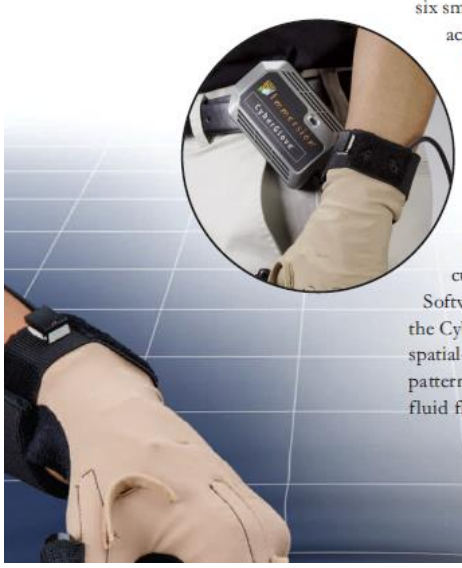
Vibrational Amplitude: 1.2 N peak-to-peak at 125 Hz (max)

Interface Unit: 3.0 x 4.55 x 1.04 in (7.62 x 11.56 x 2.64 cm)

Cable: Standard 25 ft (7.62m)

Interface: RS-232 (115.2 kbaud max)

The CyberTouch system consists of six small, lightweight vibro-tactile actuators, one on each finger and the palm of the CyberGlove data glove. Each actuator can be individually programmed to provide the desired feedback level.



The actuators can generate pulses, sustained vibration, or customized vibration patterns. Software developers can program the CyberTouch actuators to produce spatial-temporal tactile feedback patterns simulating movement or fluid flow across the hand.

About CyberGlove Systems LLC

Founded in 1990, CyberGlove Systems develops hardware and software technologies that enable users to interact with computers using their sense of touch.

For More Information

CyberGlove Systems LLC
2355 Paragon Drive, Suite D
San Jose, CA 95131
Tel: (408) 451-9463
Fax: (408) 689-4362
fyazadi@cyberglovesystems.com
www.cyberglovesystems.com

VirtualHand® is CyberGlove Systems' real-time, 3D, hand-interaction software.

2- CyberGrasp

CyberGlove Systems

CyberGrasp®

The CyberGrasp device is an innovative force-feedback system for your fingers and hand. A CyberGrasp system provides ungrounded (hand-referenced) force feedback to each finger and the palm, allowing you to feel computer-generated or tele-manipulated objects that you "reach into your computer" and grasp.

Grasp forces are produced by a network of tendons routed to the fingertips via the exoskeleton. There are five actuators, one for each finger, which can be individually programmed to prevent the user's fingers from penetrating or crushing a virtual solid object. The high-bandwidth actuators are located in a small actuator module which can be placed on the desktop. Additionally, since a CyberGrasp device does not provide grounded forces, the actuator module can also be worn in an CyberGlove Systems GraspPack™ backpack for portable operation, dramatically increasing the effective workspace.

The CyberGrasp system exerts grasp forces perpendicular to each fingertip, allowing full range of hand motion without restriction. The CyberGrasp system is fully adjustable and designed to accommodate a wide variety of hand sizes.

Specifications

- **Force: 12 N per finger (max, continuous)**
- **Weight: 16 oz (exoskeleton with out CyberGlove system)**
- **Workspace: 1 meter spherical radius from the Actuator Module**
- **CyberGlove: A CyberGlove device is required for a CyberGrasp system (22-sensor CyberGlove device recommended)**
- **Instrumentation Unit: A Force Control Unit and Actuator Module are included**
- **Interface: Ethernet**
- **CyberForce® robotic armature option recommended with the CyberGrasp system**

Hand-referenced Haptic Feedback

The CyberGrasp hardware is a lightweight, force-reflecting exoskeleton that slips over a CyberGlove® device and provides grasping force feedback to each finger relative to the palm. With the CyberGrasp force feedback system, you're able to feel the size and shape of computer-generated 3D objects you're holding in a simulated virtual world.

About CyberGlove Systems LLC

Launched in 1990, the family of CyberGlove products is the established and most sophisticated data glove solution in the marketplace. CyberGlove Systems spun off from Immersion Corporation in March 2009. The product family includes four data-glove solutions and the VirtualHand Software Development Kit (SDK). The products let users capture detailed finger, hand, and arm movement, allowing them to "reach in and manipulate" digital objects in virtual reality.

With CyberGlove products, users can more quickly prototype and animate in virtual reality thereby saving both time and money. Customers include Fortune 500 and Global 500 corporations, government agencies, and universities in the U.S., Europe, Asia, Middle East, and South America.

3- CyberForce

CyberGlove Systems

CyberForce®
Force Feedback System

The CyberForce® force feedback system enables you to rest your hand on virtual objects, feel weight and inertia while picking up a digital object, and experience surfaces and the impenetrable resistance of a virtual wall. Now you can intuitively explore simulated graphical objects and environments via the most natural interface possible — the human hand.

CyberGrasp exoskeleton, can be purchased along with the CyberGrasp system or afterwards, as an accessory.

Accurate Tracking

In addition to applying forces, the CyberForce system also provides full six-degrees-of-freedom tracking, accurately measuring three-degree-of-freedom translation and three-degree-of-freedom rotation of the hand.

Flexible Software Support

CyberForce arms come in right or left versions and can be combined as a complete dual-hand simulation and manipulation system. The CyberGlove Systems CyberGlove family of hardware products is fully supported by CyberGlove Systems' VirtualHand® SDK and VirtualHand for V5 software packages.

Specifications

Force Generation: 8.8 N max (6.6 N min)

Armature Weight: 19 lbs (8.6 kg)[1]

Workspace: 12 x 12 in (30.5 x 30.5 cm) swept through 133° with radius of 20 in (51 cm)[2]

Position Resolution: 0.0024 in (0.06 mm) max; 0.0029 in (0.073 mm) min[3]

Orientation Resolution: 0.09°

Instrumentation unit: Force Control Unit (FCU) and power supply included

Interface: Ethernet (10/100Mbps)

[1] Weight does not include CyberGrasp system.

[2] Position resolution and force generation specifications apply to the described workspace. Larger physical limits for the complete workspace are depicted on the reverse side.

[3] Based on encoder resolution and as reported by software. Actual resolution is a function of armature compliance, loading, and friction.

ANEXO VII. Código Fuente Teclado Virtual

```
/*
 * Created by SharpDevelop.
 * User: Vanessa Andreu && Toni Torronteras
 * Date: 11/06/2015
 * Time: 16:59
 *
 * To change this template use Tools | Options | Coding | Edit Standard Headers.
 */
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
using WiimoteLib;
using System.Runtime.InteropServices;
using System.Threading;
using System.Windows.Media.Effects;


namespace Teclado150611
{
    /// <summary>
    /// Description of MainForm.
    /// </summary>
    ///

    public partial class MainForm : Form
    {

        private Wiimote wm;
        public int Aceleracion=1;
        bool rumb = true;
        string TextEscrit;
```

```
        bool botonapretado = true;
        public int contador;



        public MainForm()
        {
            //
            // Inicializamos componente.
            //
            InitializeComponent();

            //Connectar Mando y control de errores
            try
            {
                wm = new Wiimote();
            }
            catch (System.IO.IOException ioe)
            {
                Console.WriteLine("No es posible ejecutar la aplicación2: " + ioe.Message);
            }

            wm.WiimoteChanged += wm_WiimoteChanged;


            try
            {
                wm.Connect();
            }
            catch (Exception wnfe)
            {
                MessageBox.Show("No es posible ejecutar la aplicación. Problemas con el establecimiento de
la conexión: " + wnfe.Message);
            }

            //wm.SetReportType(InputReport.ButtonsAccel, true);
            wm.SetReportType(InputReport.Buttons,false);
            wm.SetReportType(InputReport.IRAccel, true); //Activamos control de acelerómetro

        }

        //Funciones para controlar cambios en el mando Wii. Cambio de estado de los botones
        private void wm_WiimoteChanged(object sender, WiimoteChangedEventArgs args)
        {


            if (args.WiimoteState.ButtonState.A) //Apretar botón A del mando
            {

                if (InvokeRequired && botonapretado == true) //Invocamos la función apretar botón y actualizar
pantalla
                {
                    BeginInvoke(new apretarbotondelegate(apretarboton),
                            new object[]{
                                this.ActiveControl
                            });

                    BeginInvoke(new updatePantallaInformacionDelegate(updatePantallaInformacion),
                        new object[] {
                                this.pantalla, this.ActiveControl
                            });

                }

                botonapretado = false;
```

```
            for (int i=0; i<5; i++) //Enviamos feedback háptico al apretar botón
            {
                wm.SetRumble(rumb);

            }
            wm.SetRumble(false);



        }

        //Función para movernos por el teclado usando las teclas derecha e izquierda del mano. En
nuestro caso no la usaremos.

        /*
        if (args.WiimoteState.ButtonState.Right)
         {

            if (InvokeRequired)
            {
                BeginInvoke(new updateBotonDelegate(updateBoton),
                        new object[]{
                        this.GetNextControl(ActiveControl,true)

                        });


            }


            }
         //Función para movernos por el teclado usando las teclas derecha e izquierda del mano. En
nuestro caso no la usaremos.


         if (args.WiimoteState.ButtonState.Left)
         {
            if (InvokeRequired)
            {
                BeginInvoke(new updateBotonDelegate(updateBoton),
                        new object[]{
                            this.GetNextControl(ActiveControl, false)
                        });
                        }

            }*/

        //Mover el mando hacia la derecha
        if (args.WiimoteState.AccelState.Values.Y*Aceleracion == -1)
        {
            if (ActiveControl.TabIndex != 12) //Controlamos que el foco no esté en el dígito 12 BorrarTodo
            {

                if (InvokeRequired) //Invocamos función actualizar foco
                 {
                    BeginInvoke(new updateBotonDelegate(updateBoton),
                            new object[]{
                                this.GetNextControl(ActiveControl, true)
                            });

                // Volvemos a poner el teclado con el color original

                    colorboton();

                }

                Aceleracion=10;
```

```
            }
        }

        //Movemos el mando hacia la izquierda
        if (args.WiimoteState.AccelState.Values.Y*Aceleracion == 1)
        {

            if (ActiveControl.TabIndex != 0) //Controlamos que el foco no esté en el dígito 0
            {
                if (InvokeRequired) //Invocamos función actualizar foco
                {
                    BeginInvoke(new updateBotonDelegate(updateBoton),
                        new object[]{
                            this.GetNextControl(ActiveControl, false)
                        });

                    // Volvemos a poner el teclado con el color original
                    colorboton();
                }

                Aceleracion=10;
            }
        }

        //Activar opcion haptica pulsando el botón +
        if (args.WiimoteState.ButtonState.Plus)
        {

            rumb = true;

        }

        //Quitar opción haptica pulsando botón -
        if (args.WiimoteState.ButtonState.Minus)
        {

            rumb = false;

        }


    }

    //Funciones apretar boton
    void Button1Click(object sender, EventArgs e)
    {
        TextEscrit = TextEscrit + 1;
        pantalla.Text = TextEscrit;
        colorboton();
        button1.BackColor=System.Drawing.Color.SteelBlue;
        string path = @"C:\Users\Usuario\Documents\SharpDevelop
Projects\NuevoTeclado\Sonidos\1.WAV";
        playSound(path);

    }
    void Button2Click(object sender, EventArgs e)
    {
        TextEscrit = TextEscrit + 2;
        pantalla.Text = TextEscrit;
        colorboton();
        button2.BackColor=System.Drawing.Color.SteelBlue;
        string path = @"C:\Users\Usuario\Documents\SharpDevelop
Projects\NuevoTeclado\Sonidos\2.WAV";
        playSound(path);
```

```
        }
        void Button3Click(object sender, EventArgs e)
        {
            TextEscrit = TextEscrit + 3;
            pantalla.Text = TextEscrit;
            colorboton();
            button3.BackColor=System.Drawing.Color.SteelBlue;
            string path = @"C:\Users\Usuario\Documents\SharpDevelop
Projects\NuevoTeclado\Sonidos\3.WAV";
            playSound(path);
        }
        void Button4Click(object sender, EventArgs e)
        {
            TextEscrit = TextEscrit + 4;
            pantalla.Text = TextEscrit;
            colorboton();
            button4.BackColor=System.Drawing.Color.SteelBlue;
            string path = @"C:\Users\Usuario\Documents\SharpDevelop
Projects\NuevoTeclado\Sonidos\4.WAV";
            playSound(path);

        }
        void Button5Click(object sender, EventArgs e)
        {
            TextEscrit = TextEscrit + 5;
            pantalla.Text = TextEscrit;
            colorboton();
            button5.BackColor=System.Drawing.Color.SteelBlue;
            string path = @"C:\Users\Usuario\Documents\SharpDevelop
Projects\NuevoTeclado\Sonidos\5.WAV";
            playSound(path);

        }
        void Button6Click(object sender, EventArgs e)
        {
            TextEscrit = TextEscrit + 6;
            pantalla.Text = TextEscrit;
            colorboton();
            button6.BackColor=System.Drawing.Color.SteelBlue;
            string path = @"C:\Users\Usuario\Documents\SharpDevelop
Projects\NuevoTeclado\Sonidos\6.WAV";
            playSound(path);

        }
        void Button7Click(object sender, EventArgs e)
        {
            TextEscrit = TextEscrit + 7;
            pantalla.Text = TextEscrit;
            colorboton();
            button7.BackColor=System.Drawing.Color.SteelBlue;
            string path = @"C:\Users\Usuario\Documents\SharpDevelop
Projects\NuevoTeclado\Sonidos\7.WAV";
            playSound(path);

        }
        void Button8Click(object sender, EventArgs e)
        {

            TextEscrit=TextEscrit+8;
            pantalla.Text = TextEscrit;
            colorboton();
            button8.BackColor=System.Drawing.Color.SteelBlue;
            string path = @"C:\Users\Usuario\Documents\SharpDevelop
Projects\NuevoTeclado\Sonidos\8.WAV";
            playSound(path);

        }
```

```
    void Button9Click(object sender, EventArgs e)
    {

        TextEscrit=TextEscrit+9;
        pantalla.Text = TextEscrit;
        colorboton();
        button9.BackColor=System.Drawing.Color.SteelBlue;
        string path = @"C:\Users\Usuario\Documents\SharpDevelop
Projects\NuevoTeclado\Sonidos\9.WAV";
        playSound(path);

    }
    void Button0Click(object sender, EventArgs e)
    {

        TextEscrit = TextEscrit + 0;
        pantalla.Text = TextEscrit;
        colorboton();
        button0.BackColor=System.Drawing.Color.SteelBlue;
        string path = @"C:\Users\Usuario\Documents\SharpDevelop
Projects\NuevoTeclado\Sonidos\0.WAV";
        playSound(path);

    }

    //Funcion para borrar el último dígito introducido
    void Button11Click(object sender, EventArgs e)
    {
        //Calculamos el tamaño del string TextEscrit y le extraemos el último dígito
        if (TextEscrit != null)
        {
        TextEscrit = TextEscrit.Substring(0,TextEscrit.Length-1);

        pantalla.Text=TextEscrit;
        }

        colorboton();
        button11.BackColor=System.Drawing.Color.SteelBlue;


    }

    //Borrar Todo. Volvemos a iniciar contador y eliminamos pantalla
    void Button10Click(object sender, EventArgs e)
    {
        TextEscrit = null;
        pantalla.Text = TextEscrit;
        contador = 0;
        colorboton();
        button10.BackColor=System.Drawing.Color.SteelBlue;
    }

    //Función para reproducir audio de números
    private void playSound(string path)
    {
        System.Media.SoundPlayer player =
            new System.Media.SoundPlayer();
        player.SoundLocation = path;
        player.Load();
        player.Play();
    }

    //Invoke para colocar el Focus en el boton usando el mando
    delegate void updateBotonDelegate (Button Bot);

    public void updateBoton (Button Bot)
```

```
{

   try
   {

   if ( Bot.TabIndex <= 12 && Bot.TabIndex >= 0)
   {
      // Feedback haptico al movernos por el foco de los botones
      for (int i=0; i<15; i++)
      {
         wm.SetRumble(rumb);
      }

      wm.SetRumble(false);
   }

   Aceleracion=1;
   Bot.TabStop=true;
   Bot.Focus();

   Bot.BackColor=System.Drawing.Color.SteelBlue;
   botonapretado = true;

   }
   catch
   {

   }
}

//Invoke Apretar boton con mando

delegate void apretarbotondelegate (Button Bot);

public void apretarboton (Button Bot)
{

   Bot.PerformClick();

}

//Crono a 0 al inicializar aplicación
void MainFormLoad(object sender, EventArgs e)
{
   //Inicializar crono

   contador = 0;
   timer1.Start();
   timer1.Interval=1000;

}

//Invoke para actualizar información de la pantalla si se aprieta un botón usando el mando
delegate void updatePantallaInformacionDelegate(Label l, Button bot);

public void updatePantallaInformacion(Label l, Button bot)
{
   if (bot.TabIndex != 11)
   {
      l.Text=TextEscrit;
      botonapretado = false;
   }
}

//Funcion para inicializar de nuevo el color del teclado
public void colorboton ()
{
```

```
            button0.BackColor=System.Drawing.Color.LightBlue;
            button1.BackColor=System.Drawing.Color.LightBlue;
            button2.BackColor=System.Drawing.Color.LightBlue;
            button3.BackColor=System.Drawing.Color.LightBlue;
            button4.BackColor=System.Drawing.Color.LightBlue;
            button5.BackColor=System.Drawing.Color.LightBlue;
            button6.BackColor=System.Drawing.Color.LightBlue;
            button7.BackColor=System.Drawing.Color.LightBlue;
            button8.BackColor=System.Drawing.Color.LightBlue;
            button9.BackColor=System.Drawing.Color.LightBlue;
            button10.BackColor=System.Drawing.Color.LightBlue;
            button11.BackColor=System.Drawing.Color.LightBlue;

        }

        //Timer de control de crono
        void Timer1Tick(object sender, EventArgs e)
        {
            contador++;

            if (contador > 0 && contador <= 9)
            {
                crono.Text="0"+contador.ToString();
            }

            else
                crono.Text=contador.ToString();

        }


    }
}
```

ANEXO VIII. Código Fuente Simulador de Conducción

```
/*
 * Created by SharpDevelop.
 * User: Vanessa Andreu && Toni Torronteras
 * Date: 11/06/2015
 * Time: 15:13
 *
 * To change this template use Tools | Options | Coding | Edit Standard Headers.
 */

using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
using System.Drawing.Drawing2D;
using WiimoteLib;
using System.Text;
using System.Linq;
using System.Media;
using System.Windows.Input;


namespace mov
{
    /// <summary>
    /// Description of MainForm.
    /// </summary>
    ///
  public partial class MainForm : Form
    {

        Bitmap carretera = new Bitmap (@"F:\TFC\Imagenes mov
v6_26082015\mov\mov\carretera.png",false);

        bool rumb = true;
        public int mover;
        public int tiempo_timer =1000;
        private Wiimote wm;
        public int j, h;


        //Inicializacion del form. Incluimos la conexión del Wiimote
        //además de la función para activar las funciones de los botones y Acelerómetro
        public MainForm()
        {
            //wiimote

            try
            {
                wm = new Wiimote();
            }
            catch (System.IO.IOException ioe)
            {
                Console.WriteLine("No es posible ejecutar la aplicación: " + ioe.Message);
            }

        wm = new Wiimote();

//Se intenta establecer una conexión con el control de Wii
            try

           {

              wm.Connect();
              // Se va a añadir una función que se active cuando el mando produzca un evento
```

```
            wm.SetReportType(InputReport.Buttons, true);
            wm.WiimoteChanged += new
System.EventHandler<WiimoteChangedEventArgs>(wm_WiimoteChanged);


        }


        catch (WiimoteNotFoundException e)


        {// El Control no se encuentra enlazado con el ordenador

            MessageBox.Show(e.Message);


        }

        wm.SetReportType(InputReport.IRAccel, true);


        InitializeComponent();
        lbl_Coche.Hide();
        lbl_GO.Hide();
        lbl_GO.Text = "GAME OVER";
        lbl_win.Text= "HAS GANADO";


    }



    //Control de Wiimote como método de entrada (botonera y acelerometros
    void wm_WiimoteChanged(object sender, WiimoteChangedEventArgs e)

    {//Funcíon que se activa cuando ocurre un cambio en el estado del control de wii
        if (e.WiimoteState.ButtonState.Plus == true)
        {
            rumb= true;
        }

        if (e.WiimoteState.ButtonState.Minus == true)
        {
            rumb = false;

        }


        if (e.WiimoteState.ButtonState.Left == true)
        {
         wm.SetReportType(InputReport.IRAccel, false);
        }

        if (e.WiimoteState.ButtonState.Right == true)
        {
         wm.SetReportType(InputReport.IRAccel, true);
        }

        if (e.WiimoteState.ButtonState.Up == true)
        {
            mover = 0;
        }
        if (e.WiimoteState.ButtonState.Down == true)
        {
            mover = 1;
        }
```

```
    if (e.WiimoteState.AccelState.Values.Y > 0.55)
    {
        mover =0;

    }
if (e.WiimoteState.AccelState.Values.Y < -0.55)
    {
        mover =1;

    }

if (e.WiimoteState.AccelState.Values.Z > 0.4)
    {
        tiempo_timer =30;

    }
if (e.WiimoteState.AccelState.Values.Z < -0.4)
    {
        tiempo_timer =1;

    }

if (e.WiimoteState.ButtonState.One == true)
{

        if (InvokeRequired)
          {

           BeginInvoke(new empieza_carrera_delegate(empieza_carrera),
             new object[] {

           });

       }


}

if (e.WiimoteState.ButtonState.Two == true)
{

        if (InvokeRequired)
          {
           BeginInvoke(new para_carrera_delegate(para_carrera),
             new object[] {

           });

       }


}


}
//Empieza carrera
void Button1Click(object sender, System.EventArgs e)
{

    empieza_carrera();

}
//Finaliza la carrera
void Button2Click(object sender, EventArgs e)
{
```

```
    para_carrera();

}

//Timer que permite la transición de la imagen del bitmap /muestra el movimiento
public void Timer1Tick(object sender, System.EventArgs e)
{

    timer1.Interval = tiempo_timer;
    Graphics g = Graphics.FromImage(carretera);
    pictureBox1.Image = (Image)carretera;


    g.TranslateTransform(0,-15);
    g.DrawImage(carretera,0,0);
    Invalidate();


    //Control de "colisiones"
    Int32 intlocalx ;
    intlocalx =Convert.ToInt32( pictureBox1.PointToClient(lbl_Coche.Location).X);
    Int32  intlocaly ;
    intlocaly = Convert.ToInt32(pictureBox1.PointToClient(lbl_Coche.Location).Y);


    textBox6.Text =Convert.ToString(carretera.GetPixel(intlocalx, intlocaly).ToArgb());
    textBox1.Text= Convert.ToString(carretera.GetPixel(intlocalx+40, intlocaly+95).ToArgb());

    if (textBox6.Text != "-8355712" ||textBox1.Text != "-8355712" )
    {
        if (textBox6.Text == "-1" ||textBox1.Text == "-1")
        {
            Vibrar_Suave();
        }
        else
            Vibrar_Intensa();

    }


    if (textBox6.Text == "-8427008" || textBox1.Text == "-8427008")
    {
        Vibrar_Intensa();
        para_carrera();
    }

    if (textBox6.Text == "-65536" || textBox1.Text == "-65536")
    {
        Vibrar_Intensa();
        fin_carrera();

    }


}

//Timer que permite el movimiento de coche
public    void Timer2Tick(object sender, System.EventArgs e)
{


timer2.Interval = 200;
      switch (mover)

          {
```

```
                case 0 :
                lbl_Coche.Location = new System.Drawing.Point(this.lbl_Coche.Location.X-15,
this.lbl_Coche.Location.Y);
                mover = 0;
                break;

                case 1 :
                lbl_Coche.Location = new System.Drawing.Point(this.lbl_Coche.Location.X + 15,
this.lbl_Coche.Location.Y);
                mover = 1;
                break;

                }

        }


        //Carga de formulario
        void MainFormLoad(object sender, EventArgs e)
        {
            lbl_win.Hide();

            //MessageBox.Show("Pulsa el Botón Start o 1 para empezar la Carrera\nPulsa el Botón Stop o 2
para parar la Carrera"  ,"Start");

        }

        //Reproductor de audio (Play)
        public void playSound(string path)

        {

            System.Media.SoundPlayer player =  new System.Media.SoundPlayer();
            player.SoundLocation = path;
            player.Load();
            player.Play();

        }
        //Reproductor de audio (stop)
        public void playStop()

        {

            System.Media.SoundPlayer player =
            new System.Media.SoundPlayer();
            player.Stop();


        }



        //Invocar la funcio empieza carretera para que envia la informacion a través del hilo
        delegate void empieza_carrera_delegate ();
        void empieza_carrera ()

        {

            pictureBox1.Show();
            this.Show();
            this.Activate();
            this.Focus();
            lbl_GO.Hide();
            lbl_Coche.Show();
```

```
            for (int i = 0; i <10000;i++)
            {
            timer1.Start();


            }

            timer2.Start();
            timer3.Start();
            j=0;
            h=0;


            string path = @"F:\TFC\Imagenes mov v6_26082015\mk1.WAV";
            playSound(path);
            Vibrar();


        }


        //Invocar la funcio para carretera para que envia la informacion a través del hilo
        delegate void para_carrera_delegate();

        void para_carrera ()

        {

            pictureBox1.Hide();
            System.Drawing.Graphics formGraphics = this.CreateGraphics();
            timer1.Stop();
            timer2.Stop();
            playStop();
            formGraphics.Clear(Color.Gray);
            lbl_Coche.Hide();
            lbl_GO.Show();
            timer3.Stop();
            btn_start.Hide();
            btn_stop.Hide();


        }

        void fin_carrera ()

        {

            pictureBox1.Hide();
            System.Drawing.Graphics formGraphics = this.CreateGraphics();
            timer1.Stop();
            timer2.Stop();
            playStop();
            formGraphics.Clear(Color.Gray);
            lbl_Coche.Hide();
            lbl_GO.Hide();
            timer3.Stop();
            btn_start.Hide();
            btn_stop.Hide();
            lbl_win.Show();
                }


        //Wiimote como periférico de salida //las funciones de vibración en función con diferentes
intensidades
        void Vibrar ()
        {
```

```
            for (int i =0; i<10;i++){
            wm.SetRumble(rumb);
            wm.SetLEDs(true, true, true, true);
            }
            wm.SetRumble(false);
            wm.SetLEDs(false, false, false, false);


}


void Vibrar_Suave ()
{

            for (int i =0; i<2;i++){
            wm.SetRumble(rumb);
            wm.SetLEDs(true, true, true, true);
            }
            wm.SetRumble(false);
            wm.SetLEDs(false, false, false, false);


}

void Vibrar_Intensa ()
{

            for (int i =0; i<20;i++){
            wm.SetRumble(rumb);
            wm.SetLEDs(true, true, true, true);
            }
            wm.SetRumble(false);
            wm.SetLEDs(false, false, false, false);


}

//Timer cronometro
void Timer3Tick(object sender, System.EventArgs e)
{

   timer3.Interval = 1000;
   j++;
   if (j > 0 && j <=9)
   {
      label_segundos.Text="0" + j.ToString();
   }
   else
      label_segundos.Text= j.ToString();
   if (j>=59)
   {  j=0;
      h++;
      minutos.Text= "0" + h.ToString();


   }}}
```

ANEXO IX. Código Fuente Librería WiiMoteLib.dll

```
//////////////////////////////////////////////////////////////////////////////
//      Wiimote.cs
//      Managed Wiimote Library
//      Written by Brian Peek (http://www.brianpeek.com/)
//      for MSDN's Coding4Fun (http://msdn.microsoft.com/coding4fun/)
//      Visit http://blogs.msdn.com/coding4fun/archive/2007/03/14/1879033.aspx
//  and http://www.codeplex.com/WiimoteLib
//      for more information
//////////////////////////////////////////////////////////////////////////////

using System;
using System.Runtime.InteropServices;
using System.Diagnostics;
using System.IO;
using System.Runtime.Serialization;
using Microsoft.Win32.SafeHandles;
using System.Threading;

namespace WiimoteLib
{
        /// <summary>
        /// Implementation of Wiimote
        /// </summary>
        public class Wiimote : IDisposable
        {
                /// <summary>
                /// Event raised when Wiimote state is changed
                /// </summary>
                public event EventHandler<WiimoteChangedEventArgs> WiimoteChanged;

                /// <summary>
                /// Event raised when an extension is inserted or removed
                /// </summary>
                public event EventHandler<WiimoteExtensionChangedEventArgs>
WiimoteExtensionChanged;

                // VID = Nintendo, PID = Wiimote
                private const int VID = 0x057e;
                private const int PID = 0x0306;

                // sure, we could find this out the hard way using HID, but trust me, it's 22
                private const int REPORT_LENGTH = 22;

                // Wiimote output commands
                private enum OutputReport : byte
                {
                        LEDs            = 0x11,
                        Type            = 0x12,
                        IR                      = 0x13,
                        Status          = 0x15,
                        WriteMemory     = 0x16,
                        ReadMemory      = 0x17,
                        IR2                     = 0x1a,
                };
```

```
// Wiimote registers
private const int REGISTER_IR                                = 0x04b00030;
private const int REGISTER_IR_SENSITIVITY_1      = 0x04b00000;
private const int REGISTER_IR_SENSITIVITY_2      = 0x04b0001a;
private const int REGISTER_IR_MODE                           = 0x04b00033;

private const int REGISTER_EXTENSION_INIT                            = 0x04a40040;
private const int REGISTER_EXTENSION_TYPE                            = 0x04a400fe;
private const int REGISTER_EXTENSION_CALIBRATION     = 0x04a40020;

// read/write handle to the device
private SafeFileHandle mHandle;

// a pretty .NET stream to read/write from/to
private FileStream mStream;

// report buffer
private readonly byte[] mBuff = new byte[REPORT_LENGTH];

// read data buffer
private byte[] mReadBuff;

// address to read from
private int mAddress;

// size of requested read
private short mSize;

// current state of controller
private readonly WiimoteState mWiimoteState = new WiimoteState();

// event for read data processing
private readonly AutoResetEvent mReadDone = new AutoResetEvent(false);

// use a different method to write reports
private bool mAltWriteMethod;

/// <summary>
/// Default constructor
/// </summary>
public Wiimote()
{
}

/// <summary>
/// Connect to a Wiimote paired to the PC via Bluetooth
/// </summary>
public void Connect()
{
        int index = 0;
        bool found = false;
        Guid guid;

        // get the GUID of the HID class
        HIDImports.HidD_GetHidGuid(out guid);

        // get a handle to all devices that are part of the HID class
```

```
                        // Fun fact:  DIGCF_PRESENT worked on my machine just fine.  I reinstalled
Vista, and now it no longer finds the Wiimote with that parameter enabled...
                        IntPtr hDevInfo = HIDImports.SetupDiGetClassDevs(ref guid, null, IntPtr.Zero,
HIDImports.DIGCF_DEVICEINTERFACE);// | HIDImports.DIGCF_PRESENT);

                        // create a new interface data struct and initialize its size
                        HIDImports.SP_DEVICE_INTERFACE_DATA diData = new
HIDImports.SP_DEVICE_INTERFACE_DATA();
                        diData.cbSize = Marshal.SizeOf(diData);

                        // get a device interface to a single device (enumerate all devices)
                        while(HIDImports.SetupDiEnumDeviceInterfaces(hDevInfo, IntPtr.Zero, ref guid,
index, ref diData))
                        {
                                UInt32 size;

                                // get the buffer size for this device detail instance (returned in the size
parameter)
                                HIDImports.SetupDiGetDeviceInterfaceDetail(hDevInfo, ref diData,
IntPtr.Zero, 0, out size, IntPtr.Zero);

                                // create a detail struct and set its size
                                HIDImports.SP_DEVICE_INTERFACE_DETAIL_DATA diDetail = new
HIDImports.SP_DEVICE_INTERFACE_DETAIL_DATA();

                                // yeah, yeah...well, see, on Win x86, cbSize must be 5 for some
reason.  On x64, apparently 8 is what it wants.
                                // someday I should figure this out.  Thanks to Paul Miller on this...
                                diDetail.cbSize = (uint)(IntPtr.Size == 8 ? 8 : 5);

                                // actually get the detail struct
                                if(HIDImports.SetupDiGetDeviceInterfaceDetail(hDevInfo, ref diData,
ref diDetail, size, out size, IntPtr.Zero))
                                {
                                        Debug.WriteLine(index + " " + diDetail.DevicePath + " " +
Marshal.GetLastWin32Error());

                                        // open a read/write handle to our device using the DevicePath
returned
                                        mHandle = HIDImports.CreateFile(diDetail.DevicePath,
FileAccess.ReadWrite, FileShare.ReadWrite, IntPtr.Zero, FileMode.Open,
HIDImports.EFileAttributes.Overlapped, IntPtr.Zero);

                                        // create an attributes struct and initialize the size
                                        HIDImports.HIDD_ATTRIBUTES attrib = new
HIDImports.HIDD_ATTRIBUTES();
                                        attrib.Size = Marshal.SizeOf(attrib);

                                        // get the attributes of the current device

        if(HIDImports.HidD_GetAttributes(mHandle.DangerousGetHandle(), ref attrib))
                                        {
                                                // if the vendor and product IDs match up
                                                if(attrib.VendorID == VID && attrib.ProductID == PID)
                                                {
                                                        Debug.WriteLine("Found it!");
                                                        found = true;
```

```
                                                // create a nice .NET FileStream wrapping
the handle above
                                                mStream = new FileStream(mHandle,
FileAccess.ReadWrite, REPORT_LENGTH, true);

                                                // start an async read operation on it
                                                BeginAsyncRead();

                                                // read the calibration info from the controller
                                                try
                                                {
                                                        ReadCalibration();
                                                }
                                                catch
                                                {
                                                        // if we fail above, try the alternate
HID writes
                                                        mAltWriteMethod = true;
                                                        ReadCalibration();
                                                }

                                                // force a status check to get the state of any
extensions plugged in at startup
                                                GetStatus();

                                                break;
                                        }
                                        else
                                        {
                                                // otherwise this isn't the controller, so close
up the file handle
                                                mHandle.Close();
                                        }
                                }
                        }
                        else
                        {
                                // failed to get the detail struct
                                throw new
WiimoteException("SetupDiGetDeviceInterfaceDetail failed on index " + index);
                        }

                        // move to the next device
                        index++;
                }

                // clean up our list
                HIDImports.SetupDiDestroyDeviceInfoList(hDevInfo);

                // if we didn't find a Wiimote, throw an exception
                if(!found)
                        throw new WiimoteException("Wiimote not found in HID device list.");
        }

        /// <summary>
        /// Disconnect from the controller and stop reading data from it
        /// </summary>
```

```
		public void Disconnect()
		{
			// close up the stream and handle
			if(mStream != null)
				mStream.Close();

			if(mHandle != null)
				mHandle.Close();
		}

		/// <summary>
		/// Start reading asynchronously from the controller
		/// </summary>
	private void BeginAsyncRead()
	{
			// if the stream is valid and ready
			if(mStream != null && mStream.CanRead)
			{
				// setup the read and the callback
				byte[] buff = new byte[REPORT_LENGTH];
				mStream.BeginRead(buff, 0, REPORT_LENGTH, new
AsyncCallback(OnReadData), buff);
			}
	}

		/// <summary>
		/// Callback when data is ready to be processed
		/// </summary>
		/// <param name="ar">State information for the callback</param>
	private void OnReadData(IAsyncResult ar)
		{
			// grab the byte buffer
			byte[] buff = (byte[])ar.AsyncState;

			try
			{
				// end the current read
				mStream.EndRead(ar);

				// parse it
				if(ParseInputReport(buff))
				{
					// post an event
					if(WiimoteChanged != null)
						WiimoteChanged(this, new
WiimoteChangedEventArgs(mWiimoteState));
				}

				// start reading again
				BeginAsyncRead();
			}
			catch(OperationCanceledException)
			{
				Debug.WriteLine("OperationCanceledException");
			}
		}
```

```
		/// <summary>
		/// Parse a report sent by the Wiimote
		/// </summary>
		/// <param name="buff">Data buffer to parse</param>
		/// <returns>Returns a boolean noting whether an event needs to be posted</returns>
		private bool ParseInputReport(byte[] buff)
		{
			InputReport type = (InputReport)buff[0];

			switch(type)
			{
				case InputReport.Buttons:
					ParseButtons(buff);
					break;
				case InputReport.ButtonsAccel:
					ParseButtons(buff);
					ParseAccel(buff);
					break;
				case InputReport.IRAccel:
					ParseButtons(buff);
					ParseAccel(buff);
					ParseIR(buff);
					break;
				case InputReport.ButtonsExtension:
					ParseButtons(buff);
					ParseExtension(DecryptBuffer(buff), 4);
					break;
				case InputReport.ExtensionAccel:
					ParseButtons(buff);
					ParseAccel(buff);
					ParseExtension(DecryptBuffer(buff), 6);
					break;
				case InputReport.IRExtensionAccel:
					ParseButtons(buff);
					ParseAccel(buff);
					ParseIR(buff);
					ParseExtension(DecryptBuffer(buff), 16);
					break;
				case InputReport.Status:
					ParseButtons(buff);
					mWiimoteState.Battery = buff[6];

					// get the real LED values in case the values from SetLEDs()
somehow becomes out of sync, which really shouldn't be possible
					mWiimoteState.LEDState.LED1 = (buff[3] & 0x10) != 0;
					mWiimoteState.LEDState.LED2 = (buff[3] & 0x20) != 0;
					mWiimoteState.LEDState.LED3 = (buff[3] & 0x40) != 0;
					mWiimoteState.LEDState.LED4 = (buff[3] & 0x80) != 0;

					// extension connected?
					bool extension = (buff[3] & 0x02) != 0;
					Debug.WriteLine("Extension: " + extension);

					if(mWiimoteState.Extension != extension)
					{
						mWiimoteState.Extension = extension;

						if(extension)
```

```
                                        {
                                                // start reading again
                                                BeginAsyncRead();

                                                InitializeExtension();
                                        }
                                        else
                                                mWiimoteState.ExtensionType =
ExtensionType.None;

                                        if(WiimoteExtensionChanged != null)
                                                WiimoteExtensionChanged(this, new
WiimoteExtensionChangedEventArgs(mWiimoteState.ExtensionType, mWiimoteState.Extension));
                                }

                                break;
                        case InputReport.ReadData:
                                ParseButtons(buff);
                                ParseReadData(buff);
                                break;
                        default:
                                Debug.WriteLine("Unknown report type: " +
type.ToString("x"));
                                return false;
                }

                return true;
        }

        /// <summary>
        /// Handles setting up an extension when plugged in
        /// </summary>
        private void InitializeExtension()
        {
                WriteData(REGISTER_EXTENSION_INIT, 0x00);

                // doing a read too quickly after a write seems to kill the guitar controller
                Thread.Sleep(50);

                byte[] buff = ReadData(REGISTER_EXTENSION_TYPE, 2);

                if(buff[0] == (byte)ExtensionType.Nunchuk && buff[1] ==
(byte)ExtensionType.Nunchuk)
                        mWiimoteState.ExtensionType = ExtensionType.Nunchuk;
                else if(buff[0] == (byte)ExtensionType.ClassicController && buff[1] ==
(byte)ExtensionType.ClassicController)
                        mWiimoteState.ExtensionType = ExtensionType.ClassicController;
                else if(buff[0] == (byte)ExtensionType.ClassicController && buff[1] ==
(byte)ExtensionType.Guitar)
                        mWiimoteState.ExtensionType = ExtensionType.Guitar;
                else if(buff[0] == 0xff)        // partially inserted case...reset back to nothing
inserted
                {
                        mWiimoteState.Extension = false;
                        mWiimoteState.ExtensionType = ExtensionType.None;
                        return;
                }
```

```
                        else
                                throw new WiimoteException("Unknown extension controller found: " +
buff[0]);

                        buff = DecryptBuffer(ReadData(REGISTER_EXTENSION_CALIBRATION, 16));

                        switch(mWiimoteState.ExtensionType)
                        {
                                case ExtensionType.Nunchuk:

        mWiimoteState.NunchukState.CalibrationInfo.AccelCalibration.X0 = buff[0];

        mWiimoteState.NunchukState.CalibrationInfo.AccelCalibration.Y0 = buff[1];

        mWiimoteState.NunchukState.CalibrationInfo.AccelCalibration.Z0 = buff[2];

        mWiimoteState.NunchukState.CalibrationInfo.AccelCalibration.XG = buff[4];

        mWiimoteState.NunchukState.CalibrationInfo.AccelCalibration.YG = buff[5];

        mWiimoteState.NunchukState.CalibrationInfo.AccelCalibration.ZG = buff[6];
                                        mWiimoteState.NunchukState.CalibrationInfo.MaxX = buff[8];
                                        mWiimoteState.NunchukState.CalibrationInfo.MinX = buff[9];
                                        mWiimoteState.NunchukState.CalibrationInfo.MidX = buff[10];
                                        mWiimoteState.NunchukState.CalibrationInfo.MaxY =
buff[11];
                                        mWiimoteState.NunchukState.CalibrationInfo.MinY = buff[12];
                                        mWiimoteState.NunchukState.CalibrationInfo.MidY = buff[13];
                                        break;
                                case ExtensionType.ClassicController:
                                        mWiimoteState.ClassicControllerState.CalibrationInfo.MaxXL
= (byte)(buff[0] >> 2);
                                        mWiimoteState.ClassicControllerState.CalibrationInfo.MinXL
= (byte)(buff[1] >> 2);
                                        mWiimoteState.ClassicControllerState.CalibrationInfo.MidXL
= (byte)(buff[2] >> 2);
                                        mWiimoteState.ClassicControllerState.CalibrationInfo.MaxYL
= (byte)(buff[3] >> 2);
                                        mWiimoteState.ClassicControllerState.CalibrationInfo.MinYL
= (byte)(buff[4] >> 2);
                                        mWiimoteState.ClassicControllerState.CalibrationInfo.MidYL
= (byte)(buff[5] >> 2);

                                        mWiimoteState.ClassicControllerState.CalibrationInfo.MaxXR
= (byte)(buff[6] >> 3);
                                        mWiimoteState.ClassicControllerState.CalibrationInfo.MinXR
= (byte)(buff[7] >> 3);
                                        mWiimoteState.ClassicControllerState.CalibrationInfo.MidXR
= (byte)(buff[8] >> 3);
                                        mWiimoteState.ClassicControllerState.CalibrationInfo.MaxYR
= (byte)(buff[9] >> 3);
                                        mWiimoteState.ClassicControllerState.CalibrationInfo.MinYR
= (byte)(buff[10] >> 3);
                                        mWiimoteState.ClassicControllerState.CalibrationInfo.MidYR
= (byte)(buff[11] >> 3);

                                        // this doesn't seem right...
```

```
//
            mWiimoteState.ClassicControllerState.AccelCalibrationInfo.MinTriggerL = (byte)(buff[12] >> 3);
//
            mWiimoteState.ClassicControllerState.AccelCalibrationInfo.MaxTriggerL = (byte)(buff[14] >> 3);
//
            mWiimoteState.ClassicControllerState.AccelCalibrationInfo.MinTriggerR = (byte)(buff[13] >> 3);
//
            mWiimoteState.ClassicControllerState.AccelCalibrationInfo.MaxTriggerR = (byte)(buff[15] >> 3);

            mWiimoteState.ClassicControllerState.CalibrationInfo.MinTriggerL = 0;

            mWiimoteState.ClassicControllerState.CalibrationInfo.MaxTriggerL = 31;

            mWiimoteState.ClassicControllerState.CalibrationInfo.MinTriggerR = 0;

            mWiimoteState.ClassicControllerState.CalibrationInfo.MaxTriggerR = 31;
                                    break;
                                case ExtensionType.Guitar:
                                    // there appears to be no calibration data returned by the
guitar controller
                                    break;
                            }
                        }

                        /// <summary>
                        /// Decrypts data sent from the extension to the Wiimote
                        /// </summary>
                        /// <param name="buff">Data buffer</param>
                        /// <returns>Byte array containing decoded data</returns>
                        private byte[] DecryptBuffer(byte[] buff)
                        {
                            for(int i = 0; i < buff.Length; i++)
                                buff[i] = (byte)(((buff[i] ^ 0x17) + 0x17) & 0xff);

                            return buff;
                        }

                        /// <summary>
                        /// Parses a standard button report into the ButtonState struct
                        /// </summary>
                        /// <param name="buff">Data buffer</param>
                        private void ParseButtons(byte[] buff)
                        {
                            mWiimoteState.ButtonState.A     = (buff[2] & 0x08) != 0;
                            mWiimoteState.ButtonState.B     = (buff[2] & 0x04) != 0;
                            mWiimoteState.ButtonState.Minus = (buff[2] & 0x10) != 0;
                            mWiimoteState.ButtonState.Home  = (buff[2] & 0x80) != 0;
                            mWiimoteState.ButtonState.Plus  = (buff[1] & 0x10) != 0;
                            mWiimoteState.ButtonState.One   = (buff[2] & 0x02) != 0;
                            mWiimoteState.ButtonState.Two   = (buff[2] & 0x01) != 0;
                            mWiimoteState.ButtonState.Up    = (buff[1] & 0x08) != 0;
                            mWiimoteState.ButtonState.Down  = (buff[1] & 0x04) != 0;
                            mWiimoteState.ButtonState.Left  = (buff[1] & 0x01) != 0;
                            mWiimoteState.ButtonState.Right = (buff[1] & 0x02) != 0;
                        }

                        /// <summary>
```

```
		/// Parse accelerometer data
		/// </summary>
		/// <param name="buff">Data buffer</param>
		private void ParseAccel(byte[] buff)
		{
			mWiimoteState.AccelState.RawValues.X = buff[3];
			mWiimoteState.AccelState.RawValues.Y = buff[4];
			mWiimoteState.AccelState.RawValues.Z = buff[5];

			mWiimoteState.AccelState.Values.X =
(float)((float)mWiimoteState.AccelState.RawValues.X - mWiimoteState.AccelCalibrationInfo.X0) /

	((float)mWiimoteState.AccelCalibrationInfo.XG - mWiimoteState.AccelCalibrationInfo.X0);
			mWiimoteState.AccelState.Values.Y =
(float)((float)mWiimoteState.AccelState.RawValues.Y - mWiimoteState.AccelCalibrationInfo.Y0) /

	((float)mWiimoteState.AccelCalibrationInfo.YG - mWiimoteState.AccelCalibrationInfo.Y0);
			mWiimoteState.AccelState.Values.Z =
(float)((float)mWiimoteState.AccelState.RawValues.Z - mWiimoteState.AccelCalibrationInfo.Z0) /

	((float)mWiimoteState.AccelCalibrationInfo.ZG - mWiimoteState.AccelCalibrationInfo.Z0);
		}

		/// <summary>
		/// Parse IR data from report
		/// </summary>
		/// <param name="buff">Data buffer</param>
		private void ParseIR(byte[] buff)
		{
			mWiimoteState.IRState.IRSensors[0].RawPosition.X = buff[6]  | ((buff[8] >> 4) &
0x03) << 8;
			mWiimoteState.IRState.IRSensors[0].RawPosition.Y = buff[7]  | ((buff[8] >> 6) &
0x03) << 8;

			switch(mWiimoteState.IRState.Mode)
			{
				case IRMode.Basic:
					mWiimoteState.IRState.IRSensors[1].RawPosition.X = buff[9]
| ((buff[8] >> 0) & 0x03) << 8;
					mWiimoteState.IRState.IRSensors[1].RawPosition.Y =
buff[10] | ((buff[8] >> 2) & 0x03) << 8;

					mWiimoteState.IRState.IRSensors[0].Size = 0x00;
					mWiimoteState.IRState.IRSensors[1].Size = 0x00;

					mWiimoteState.IRState.IRSensors[0].Found = !(buff[6] == 0xff
&& buff[7] == 0xff);
					mWiimoteState.IRState.IRSensors[1].Found = !(buff[9] == 0xff
&& buff[10] == 0xff);
					break;
				case IRMode.Extended:
					mWiimoteState.IRState.IRSensors[1].RawPosition.X = buff[9]
| ((buff[11] >> 4) & 0x03) << 8;
					mWiimoteState.IRState.IRSensors[1].RawPosition.Y =
buff[10] | ((buff[11] >> 6) & 0x03) << 8;
					mWiimoteState.IRState.IRSensors[2].RawPosition.X =
buff[12] | ((buff[14] >> 4) & 0x03) << 8;
```

```
                                        mWiimoteState.IRState.IRSensors[2].RawPosition.Y =
buff[13] | ((buff[14] >> 6) & 0x03) << 8;
                                        mWiimoteState.IRState.IRSensors[3].RawPosition.X =
buff[15] | ((buff[17] >> 4) & 0x03) << 8;
                                        mWiimoteState.IRState.IRSensors[3].RawPosition.Y =
buff[16] | ((buff[17] >> 6) & 0x03) << 8;

                                        mWiimoteState.IRState.IRSensors[0].Size = buff[8] & 0x0f;
                                        mWiimoteState.IRState.IRSensors[1].Size = buff[11] & 0x0f;
                                        mWiimoteState.IRState.IRSensors[2].Size = buff[14] & 0x0f;
                                        mWiimoteState.IRState.IRSensors[3].Size = buff[17] & 0x0f;

                                        mWiimoteState.IRState.IRSensors[0].Found = !(buff[6] == 0xff
&& buff[7] == 0xff && buff[8] == 0xff);
                                        mWiimoteState.IRState.IRSensors[1].Found = !(buff[9] == 0xff
&& buff[10] == 0xff && buff[11] == 0xff);
                                        mWiimoteState.IRState.IRSensors[2].Found = !(buff[12] ==
0xff && buff[13] == 0xff && buff[14] == 0xff);
                                        mWiimoteState.IRState.IRSensors[3].Found = !(buff[15] ==
0xff && buff[16] == 0xff && buff[17] == 0xff);
                                        break;
                                }

                                mWiimoteState.IRState.IRSensors[0].Position.X =
(float)(mWiimoteState.IRState.IRSensors[0].RawPosition.X / 1023.5f);
                                mWiimoteState.IRState.IRSensors[1].Position.X =
(float)(mWiimoteState.IRState.IRSensors[1].RawPosition.X / 1023.5f);
                                mWiimoteState.IRState.IRSensors[2].Position.X =
(float)(mWiimoteState.IRState.IRSensors[2].RawPosition.X / 1023.5f);
                                mWiimoteState.IRState.IRSensors[3].Position.X =
(float)(mWiimoteState.IRState.IRSensors[3].RawPosition.X / 1023.5f);

                                mWiimoteState.IRState.IRSensors[0].Position.Y =
(float)(mWiimoteState.IRState.IRSensors[0].RawPosition.Y / 767.5f);
                                mWiimoteState.IRState.IRSensors[1].Position.Y =
(float)(mWiimoteState.IRState.IRSensors[1].RawPosition.Y / 767.5f);
                                mWiimoteState.IRState.IRSensors[2].Position.Y =
(float)(mWiimoteState.IRState.IRSensors[2].RawPosition.Y / 767.5f);
                                mWiimoteState.IRState.IRSensors[3].Position.Y =
(float)(mWiimoteState.IRState.IRSensors[3].RawPosition.Y / 767.5f);

                                if(mWiimoteState.IRState.IRSensors[0].Found &&
mWiimoteState.IRState.IRSensors[1].Found)
                                {
                                        mWiimoteState.IRState.RawMidpoint.X =
(mWiimoteState.IRState.IRSensors[1].RawPosition.X +
mWiimoteState.IRState.IRSensors[0].RawPosition.X) / 2;
                                        mWiimoteState.IRState.RawMidpoint.Y =
(mWiimoteState.IRState.IRSensors[1].RawPosition.Y +
mWiimoteState.IRState.IRSensors[0].RawPosition.Y) / 2;

                                        mWiimoteState.IRState.Midpoint.X =
(mWiimoteState.IRState.IRSensors[1].Position.X + mWiimoteState.IRState.IRSensors[0].Position.X) / 2.0f;
                                        mWiimoteState.IRState.Midpoint.Y =
(mWiimoteState.IRState.IRSensors[1].Position.Y + mWiimoteState.IRState.IRSensors[0].Position.Y) / 2.0f;
                                }
                                else
```

```
                                mWiimoteState.IRState.Midpoint.X =
mWiimoteState.IRState.Midpoint.Y = 0.0f;
                }

                /// <summary>
                /// Parse data from an extension controller
                /// </summary>
                /// <param name="buff">Data buffer</param>
                /// <param name="offset">Offset into data buffer</param>
                private void ParseExtension(byte[] buff, int offset)
                {
                        switch(mWiimoteState.ExtensionType)
                        {
                                case ExtensionType.Nunchuk:
                                        mWiimoteState.NunchukState.RawJoystick.X = buff[offset];
                                        mWiimoteState.NunchukState.RawJoystick.Y = buff[offset +
1];
                                        mWiimoteState.NunchukState.AccelState.RawValues.X =
buff[offset + 2];
                                        mWiimoteState.NunchukState.AccelState.RawValues.Y =
buff[offset + 3];
                                        mWiimoteState.NunchukState.AccelState.RawValues.Z =
buff[offset + 4];

                                        mWiimoteState.NunchukState.C = (buff[offset + 5] & 0x02) ==
0;
                                        mWiimoteState.NunchukState.Z = (buff[offset + 5] & 0x01) ==
0;

                                        mWiimoteState.NunchukState.AccelState.Values.X =
(float)((float)mWiimoteState.NunchukState.AccelState.RawValues.X -
mWiimoteState.NunchukState.CalibrationInfo.AccelCalibration.X0) /

        ((float)mWiimoteState.NunchukState.CalibrationInfo.AccelCalibration.XG -
mWiimoteState.NunchukState.CalibrationInfo.AccelCalibration.X0);
                                        mWiimoteState.NunchukState.AccelState.Values.Y =
(float)((float)mWiimoteState.NunchukState.AccelState.RawValues.Y -
mWiimoteState.NunchukState.CalibrationInfo.AccelCalibration.Y0) /

        ((float)mWiimoteState.NunchukState.CalibrationInfo.AccelCalibration.YG -
mWiimoteState.NunchukState.CalibrationInfo.AccelCalibration.Y0);
                                        mWiimoteState.NunchukState.AccelState.Values.Z =
(float)((float)mWiimoteState.NunchukState.AccelState.RawValues.Z -
mWiimoteState.NunchukState.CalibrationInfo.AccelCalibration.Z0) /

        ((float)mWiimoteState.NunchukState.CalibrationInfo.AccelCalibration.ZG -
mWiimoteState.NunchukState.CalibrationInfo.AccelCalibration.Z0);

                                        if(mWiimoteState.NunchukState.CalibrationInfo.MaxX !=
0x00)
                                                mWiimoteState.NunchukState.Joystick.X =
(float)((float)mWiimoteState.NunchukState.RawJoystick.X -
mWiimoteState.NunchukState.CalibrationInfo.MidX) /

        ((float)mWiimoteState.NunchukState.CalibrationInfo.MaxX -
mWiimoteState.NunchukState.CalibrationInfo.MinX);
```

```
                                        if(mWiimoteState.NunchukState.CalibrationInfo.MaxY !=
0x00)
                                            mWiimoteState.NunchukState.Joystick.Y =
(float)((float)mWiimoteState.NunchukState.RawJoystick.Y -
mWiimoteState.NunchukState.CalibrationInfo.MidY) /

        ((float)mWiimoteState.NunchukState.CalibrationInfo.MaxY -
mWiimoteState.NunchukState.CalibrationInfo.MinY);

                                        break;

                                    case ExtensionType.ClassicController:
                                        mWiimoteState.ClassicControllerState.RawJoystickL.X =
(byte)(buff[offset] & 0x3f);
                                        mWiimoteState.ClassicControllerState.RawJoystickL.Y =
(byte)(buff[offset + 1] & 0x3f);
                                        mWiimoteState.ClassicControllerState.RawJoystickR.X =
(byte)((buff[offset + 2] >> 7) | (buff[offset + 1] & 0xc0) >> 5 | (buff[offset] & 0xc0) >> 3);
                                        mWiimoteState.ClassicControllerState.RawJoystickR.Y =
(byte)(buff[offset + 2] & 0x1f);

                                        mWiimoteState.ClassicControllerState.RawTriggerL =
(byte)(((buff[offset + 2] & 0x60) >> 2) | (buff[offset + 3] >> 5));
                                        mWiimoteState.ClassicControllerState.RawTriggerR =
(byte)(buff[offset + 3] & 0x1f);

                                        mWiimoteState.ClassicControllerState.ButtonState.TriggerR
        = (buff[offset + 4] & 0x02) == 0;
                                        mWiimoteState.ClassicControllerState.ButtonState.Plus
        = (buff[offset + 4] & 0x04) == 0;
                                        mWiimoteState.ClassicControllerState.ButtonState.Home
        = (buff[offset + 4] & 0x08) == 0;
                                        mWiimoteState.ClassicControllerState.ButtonState.Minus
        = (buff[offset + 4] & 0x10) == 0;
                                        mWiimoteState.ClassicControllerState.ButtonState.TriggerL
        = (buff[offset + 4] & 0x20) == 0;
                                        mWiimoteState.ClassicControllerState.ButtonState.Down
        = (buff[offset + 4] & 0x40) == 0;
                                        mWiimoteState.ClassicControllerState.ButtonState.Right
        = (buff[offset + 4] & 0x80) == 0;

                                        mWiimoteState.ClassicControllerState.ButtonState.Up
                = (buff[offset + 5] & 0x01) == 0;
                                        mWiimoteState.ClassicControllerState.ButtonState.Left
        = (buff[offset + 5] & 0x02) == 0;
                                        mWiimoteState.ClassicControllerState.ButtonState.ZR
                = (buff[offset + 5] & 0x04) == 0;
                                        mWiimoteState.ClassicControllerState.ButtonState.X
        = (buff[offset + 5] & 0x08) == 0;
                                        mWiimoteState.ClassicControllerState.ButtonState.A
        = (buff[offset + 5] & 0x10) == 0;
                                        mWiimoteState.ClassicControllerState.ButtonState.Y
        = (buff[offset + 5] & 0x20) == 0;
                                        mWiimoteState.ClassicControllerState.ButtonState.B
        = (buff[offset + 5] & 0x40) == 0;
                                        mWiimoteState.ClassicControllerState.ButtonState.ZL
                = (buff[offset + 5] & 0x80) == 0;
```

```
        if(mWiimoteState.ClassicControllerState.CalibrationInfo.MaxXL != 0x00)
                                                    mWiimoteState.ClassicControllerState.JoystickL.X =
(float)((float)mWiimoteState.ClassicControllerState.RawJoystickL.X -
mWiimoteState.ClassicControllerState.CalibrationInfo.MidXL) /

        (float)(mWiimoteState.ClassicControllerState.CalibrationInfo.MaxXL -
mWiimoteState.ClassicControllerState.CalibrationInfo.MinXL);


        if(mWiimoteState.ClassicControllerState.CalibrationInfo.MaxYL != 0x00)
                                                    mWiimoteState.ClassicControllerState.JoystickL.Y =
(float)((float)mWiimoteState.ClassicControllerState.RawJoystickL.Y -
mWiimoteState.ClassicControllerState.CalibrationInfo.MidYL) /

        (float)(mWiimoteState.ClassicControllerState.CalibrationInfo.MaxYL -
mWiimoteState.ClassicControllerState.CalibrationInfo.MinYL);


        if(mWiimoteState.ClassicControllerState.CalibrationInfo.MaxXR != 0x00)
                                                    mWiimoteState.ClassicControllerState.JoystickR.X =
(float)((float)mWiimoteState.ClassicControllerState.RawJoystickR.X -
mWiimoteState.ClassicControllerState.CalibrationInfo.MidXR) /

        (float)(mWiimoteState.ClassicControllerState.CalibrationInfo.MaxXR -
mWiimoteState.ClassicControllerState.CalibrationInfo.MinXR);


        if(mWiimoteState.ClassicControllerState.CalibrationInfo.MaxYR != 0x00)
                                                    mWiimoteState.ClassicControllerState.JoystickR.Y =
(float)((float)mWiimoteState.ClassicControllerState.RawJoystickR.Y -
mWiimoteState.ClassicControllerState.CalibrationInfo.MidYR) /

        (float)(mWiimoteState.ClassicControllerState.CalibrationInfo.MaxYR -
mWiimoteState.ClassicControllerState.CalibrationInfo.MinYR);


        if(mWiimoteState.ClassicControllerState.CalibrationInfo.MaxTriggerL != 0x00)
                                                    mWiimoteState.ClassicControllerState.TriggerL =
(mWiimoteState.ClassicControllerState.RawTriggerL) /

        (float)(mWiimoteState.ClassicControllerState.CalibrationInfo.MaxTriggerL -
mWiimoteState.ClassicControllerState.CalibrationInfo.MinTriggerL);


        if(mWiimoteState.ClassicControllerState.CalibrationInfo.MaxTriggerR != 0x00)
                                                    mWiimoteState.ClassicControllerState.TriggerR =
(mWiimoteState.ClassicControllerState.RawTriggerR) /

        (float)(mWiimoteState.ClassicControllerState.CalibrationInfo.MaxTriggerR -
mWiimoteState.ClassicControllerState.CalibrationInfo.MinTriggerR);
                                            break;

                                    case ExtensionType.Guitar:
                                        mWiimoteState.GuitarState.ButtonState.Plus                   =
(buff[offset + 4] & 0x04) == 0;
```

```
                                mWiimoteState.GuitarState.ButtonState.Minus          =
(buff[offset + 4] & 0x10) == 0;
                                mWiimoteState.GuitarState.ButtonState.StrumDown     =
(buff[offset + 4] & 0x40) == 0;

                                mWiimoteState.GuitarState.ButtonState.StrumUp  = (buff[offset + 5]
& 0x01) == 0;
                                mWiimoteState.GuitarState.ButtonState.Yellow     = (buff[offset + 5]
& 0x08) == 0;
                                mWiimoteState.GuitarState.ButtonState.Green          =
(buff[offset + 5] & 0x10) == 0;
                                mWiimoteState.GuitarState.ButtonState.Blue           =
(buff[offset + 5] & 0x20) == 0;
                                mWiimoteState.GuitarState.ButtonState.Red            =
(buff[offset + 5] & 0x40) == 0;
                                mWiimoteState.GuitarState.ButtonState.Orange    = (buff[offset + 5]
& 0x80) == 0;

                                    // it appears the joystick values are only 6 bits
                                    mWiimoteState.GuitarState.RawJoystick.X
        = (buff[offset + 0] & 0x3f);
                                    mWiimoteState.GuitarState.RawJoystick.Y
        = (buff[offset + 1] & 0x3f);

                                    // and the whammy bar is only 4 bits
                                    mWiimoteState.GuitarState.RawWhammyBar
        = (byte)(buff[offset + 3] & 0x0f);

                                    mWiimoteState.GuitarState.Joystick.X
        = (float)(mWiimoteState.GuitarState.RawJoystick.X - 0x1f) / 0x3f;        // not fully accurate, but
close
                                    mWiimoteState.GuitarState.Joystick.Y
        = (float)(mWiimoteState.GuitarState.RawJoystick.Y - 0x1f) / 0x3f;        // not fully accurate, but
close
                                    mWiimoteState.GuitarState.WhammyBar
        = (float)(mWiimoteState.GuitarState.RawWhammyBar) / 0x0a;            // seems like there are 10
positions?
                                break;
                        }
                }

                /// <summary>
                /// Parse data returned from a read report
                /// </summary>
                /// <param name="buff">Data buffer</param>
                private void ParseReadData(byte[] buff)
                {
                        if((buff[3] & 0x08) != 0)
                                throw new WiimoteException("Error reading data from Wiimote: Bytes
do not exist.");

                        if((buff[3] & 0x07) != 0)
                                throw new WiimoteException("Error reading data from Wiimote:
Attempt to read from write-only registers.");

                        // get our size and offset from the report
                        int size = (buff[3] >> 4) + 1;
```

```
                int offset = (buff[4] << 8 | buff[5]);

                // add it to the buffer
                Array.Copy(buff, 6, mReadBuff, offset - mAddress, size);

                // if we've read it all, set the event
                if(mAddress + mSize == offset + size)
                        mReadDone.Set();
        }

        /// <summary>
        /// Returns whether rumble is currently enabled.
        /// </summary>
        /// <returns>Byte indicating true (0x01) or false (0x00)</returns>
        private byte GetRumbleBit()
        {
                return (byte)(mWiimoteState.Rumble ? 0x01 : 0x00);
        }

        /// <summary>
        /// Read calibration information stored on Wiimote
        /// </summary>
        private void ReadCalibration()
        {
                // this appears to change the report type to 0x31
                byte[] buff = ReadData(0x0016, 7);

                mWiimoteState.AccelCalibrationInfo.X0 = buff[0];
                mWiimoteState.AccelCalibrationInfo.Y0 = buff[1];
                mWiimoteState.AccelCalibrationInfo.Z0 = buff[2];
                mWiimoteState.AccelCalibrationInfo.XG = buff[4];
                mWiimoteState.AccelCalibrationInfo.YG = buff[5];
                mWiimoteState.AccelCalibrationInfo.ZG = buff[6];
        }

        /// <summary>
        /// Set Wiimote reporting mode (if using an IR report type, IR sensitivity is set to
WiiLevel3)
        /// </summary>
        /// <param name="type">Report type</param>
        /// <param name="continuous">Continuous data</param>
        public void SetReportType(InputReport type, bool continuous)
        {
                SetReportType(type, IRSensitivity.Maximum, continuous);
        }

        /// <summary>
        /// Set Wiimote reporting mode
        /// </summary>
        /// <param name="type">Report type</param>
        /// <param name="irSensitivity">IR sensitivity</param>
        /// <param name="continuous">Continuous data</param>
        public void SetReportType(InputReport type, IRSensitivity irSensitivity, bool continuous)
        {
                switch(type)
                {
                        case InputReport.IRAccel:
                                EnableIR(IRMode.Extended, irSensitivity);
```

```
                                        break;
                                case InputReport.IRExtensionAccel:
                                        EnableIR(IRMode.Basic, irSensitivity);
                                        break;
                                default:
                                        DisableIR();
                                        break;
                        }

                        ClearReport();
                        mBuff[0] = (byte)OutputReport.Type;
                        mBuff[1] = (byte)((continuous ? 0x04 : 0x00) | (byte)(mWiimoteState.Rumble ?
0x01 : 0x00));
                        mBuff[2] = (byte)type;

                        WriteReport();
                }

                /// <summary>
                /// Set the LEDs on the Wiimote
                /// </summary>
                /// <param name="led1">LED 1</param>
                /// <param name="led2">LED 2</param>
                /// <param name="led3">LED 3</param>
                /// <param name="led4">LED 4</param>
                public void SetLEDs(bool led1, bool led2, bool led3, bool led4)
                {
                        mWiimoteState.LEDState.LED1 = led1;
                        mWiimoteState.LEDState.LED2 = led2;
                        mWiimoteState.LEDState.LED3 = led3;
                        mWiimoteState.LEDState.LED4 = led4;

                        ClearReport();

                        mBuff[0] = (byte)OutputReport.LEDs;
                        mBuff[1] =      (byte)(
                                                (led1 ? 0x10 : 0x00) |
                                                (led2 ? 0x20 : 0x00) |
                                                (led3 ? 0x40 : 0x00) |
                                                (led4 ? 0x80 : 0x00) |
                                                GetRumbleBit());

                        WriteReport();
                }

                /// <summary>
                /// Set the LEDs on the Wiimote
                /// </summary>
                /// <param name="leds">The value to be lit up in base2 on the Wiimote</param>
                public void SetLEDs(int leds)
                {
                        mWiimoteState.LEDState.LED1 = (leds & 0x01) > 0;
                        mWiimoteState.LEDState.LED2 = (leds & 0x02) > 0;
                        mWiimoteState.LEDState.LED3 = (leds & 0x04) > 0;
                        mWiimoteState.LEDState.LED4 = (leds & 0x08) > 0;

                        ClearReport();
```

```
                mBuff[0] = (byte)OutputReport.LEDs;
                mBuff[1] =      (byte)(
                                        ((leds & 0x01) > 0 ? 0x10 : 0x00) |
                                        ((leds & 0x02) > 0 ? 0x20 : 0x00) |
                                        ((leds & 0x04) > 0 ? 0x40 : 0x00) |
                                        ((leds & 0x08) > 0 ? 0x80 : 0x00) |
                                        GetRumbleBit());

                WriteReport();
            }

            /// <summary>
            /// Toggle rumble
            /// </summary>
            /// <param name="on">On or off</param>
            public void SetRumble(bool on)
            {
                mWiimoteState.Rumble = on;

                // the LED report also handles rumble
                SetLEDs(mWiimoteState.LEDState.LED1,
                        mWiimoteState.LEDState.LED2,
                        mWiimoteState.LEDState.LED3,
                        mWiimoteState.LEDState.LED4);
            }

            /// <summary>
            /// Retrieve the current status of the Wiimote and extensions.  Replaces
GetBatteryLevel() since it was poorly named.
            /// </summary>
            public void GetStatus()
            {
                ClearReport();

                mBuff[0] = (byte)OutputReport.Status;
                mBuff[1] = GetRumbleBit();

                WriteReport();
            }

            /// <summary>
            /// Turn on the IR sensor
            /// </summary>
            /// <param name="mode">The data report mode</param>
            /// <param name="irSensitivity">IR sensitivity</param>
            private void EnableIR(IRMode mode, IRSensitivity irSensitivity)
            {
                mWiimoteState.IRState.Mode = mode;

                ClearReport();
                mBuff[0] = (byte)OutputReport.IR;
                mBuff[1] = (byte)(0x04 | GetRumbleBit());
                WriteReport();

                ClearReport();
                mBuff[0] = (byte)OutputReport.IR2;
                mBuff[1] = (byte)(0x04 | GetRumbleBit());
```

```
                        WriteReport();

                        WriteData(REGISTER_IR, 0x08);
                        switch(irSensitivity)
                        {
                                case IRSensitivity.WiiLevel1:
                                        WriteData(REGISTER_IR_SENSITIVITY_1, 9, new byte[]
{0x02, 0x00, 0x00, 0x71, 0x01, 0x00, 0x64, 0x00, 0xfe});
                                        WriteData(REGISTER_IR_SENSITIVITY_2, 2, new byte[]
{0xfd, 0x05});
                                        break;
                                case IRSensitivity.WiiLevel2:
                                        WriteData(REGISTER_IR_SENSITIVITY_1, 9, new byte[]
{0x02, 0x00, 0x00, 0x71, 0x01, 0x00, 0x96, 0x00, 0xb4});
                                        WriteData(REGISTER_IR_SENSITIVITY_2, 2, new byte[]
{0xb3, 0x04});
                                        break;
                                case IRSensitivity.WiiLevel3:
                                        WriteData(REGISTER_IR_SENSITIVITY_1, 9, new byte[]
{0x02, 0x00, 0x00, 0x71, 0x01, 0x00, 0xaa, 0x00, 0x64});
                                        WriteData(REGISTER_IR_SENSITIVITY_2, 2, new byte[]
{0x63, 0x03});
                                        break;
                                case IRSensitivity.WiiLevel4:
                                        WriteData(REGISTER_IR_SENSITIVITY_1, 9, new byte[]
{0x02, 0x00, 0x00, 0x71, 0x01, 0x00, 0xc8, 0x00, 0x36});
                                        WriteData(REGISTER_IR_SENSITIVITY_2, 2, new byte[]
{0x35, 0x03});
                                        break;
                                case IRSensitivity.WiiLevel5:
                                        WriteData(REGISTER_IR_SENSITIVITY_1, 9, new byte[]
{0x07, 0x00, 0x00, 0x71, 0x01, 0x00, 0x72, 0x00, 0x20});
                                        WriteData(REGISTER_IR_SENSITIVITY_2, 2, new byte[]
{0x1, 0x03});
                                        break;
                                case IRSensitivity.Maximum:
                                        WriteData(REGISTER_IR_SENSITIVITY_1, 9, new byte[]
{0x02, 0x00, 0x00, 0x71, 0x01, 0x00, 0x90, 0x00, 0x41});
                                        WriteData(REGISTER_IR_SENSITIVITY_2, 2, new byte[]
{0x40, 0x00});
                                        break;
                                default:
                                        throw new ArgumentOutOfRangeException("irSensitivity");
                        }
                        WriteData(REGISTER_IR_MODE, (byte)mode);
                        WriteData(REGISTER_IR, 0x08);
                }

                /// <summary>
                /// Disable the IR sensor
                /// </summary>
                private void DisableIR()
                {
                        mWiimoteState.IRState.Mode = IRMode.Off;

                        ClearReport();
                        mBuff[0] = (byte)OutputReport.IR;
```

```
                mBuff[1] = GetRumbleBit();
                WriteReport();

                ClearReport();
                mBuff[0] = (byte)OutputReport.IR2;
                mBuff[1] = GetRumbleBit();
                WriteReport();
            }

            /// <summary>
            /// Initialize the report data buffer
            /// </summary>
            private void ClearReport()
            {
                Array.Clear(mBuff, 0, REPORT_LENGTH);
            }

            /// <summary>
            /// Write a report to the Wiimote
            /// </summary>
            private void WriteReport()
            {
                if(mAltWriteMethod)

        HIDImports.HidD_SetOutputReport(this.mHandle.DangerousGetHandle(), mBuff,
(uint)mBuff.Length);
                else if(mStream != null)
                    mStream.Write(mBuff, 0, REPORT_LENGTH);

                Thread.Sleep(100);
            }

            /// <summary>
            /// Read data or register from Wiimote
            /// </summary>
            /// <param name="address">Address to read</param>
            /// <param name="size">Length to read</param>
            /// <returns>Data buffer</returns>
            public byte[] ReadData(int address, short size)
            {
                ClearReport();

                mReadBuff = new byte[size];
                mAddress = address & 0xffff;
                mSize = size;

                mBuff[0] = (byte)OutputReport.ReadMemory;
                mBuff[1] = (byte)(((address & 0xff000000) >> 24) | GetRumbleBit());
                mBuff[2] = (byte)((address & 0x00ff0000)  >> 16);
                mBuff[3] = (byte)((address & 0x0000ff00)  >>  8);
                mBuff[4] = (byte)(address & 0x000000ff);

                mBuff[5] = (byte)((size & 0xff00) >> 8);
                mBuff[6] = (byte)(size & 0xff);

                WriteReport();

                if(!mReadDone.WaitOne(1000, false))
```

```
                    throw new WiimoteException("Error reading data from Wiimote...is it
connected?");

                return mReadBuff;
            }

            /// <summary>
            /// Write a single byte to the Wiimote
            /// </summary>
            /// <param name="address">Address to write</param>
            /// <param name="data">Byte to write</param>
            public void WriteData(int address, byte data)
            {
                WriteData(address, 1, new byte[] { data });
            }

            /// <summary>
            /// Write a byte array to a specified address
            /// </summary>
            /// <param name="address">Address to write</param>
            /// <param name="size">Length of buffer</param>
            /// <param name="buff">Data buffer</param>

            public void WriteData(int address, byte size, byte[] buff)
            {
                ClearReport();

                mBuff[0] = (byte)OutputReport.WriteMemory;
                mBuff[1] = (byte)(((address & 0xff000000) >> 24) | GetRumbleBit());
                mBuff[2] = (byte)((address & 0x00ff0000)  >> 16);
                mBuff[3] = (byte)((address & 0x0000ff00)  >>  8);
                mBuff[4] = (byte)(address & 0x000000ff);
                mBuff[5] = size;
                Array.Copy(buff, 0, mBuff, 6, size);

                WriteReport();

                Thread.Sleep(100);
            }

            /// <summary>
            /// Current Wiimote state
            /// </summary>
            public WiimoteState WiimoteState
            {
                get { return mWiimoteState; }
            }

            #region IDisposable Members

            /// <summary>
            /// Dispose Wiimote
            /// </summary>
            public void Dispose()
            {
                Dispose(true);
                GC.SuppressFinalize(this);
```

```
                }

                /// <summary>
                /// Dispose wiimote
                /// </summary>
                /// <param name="disposing">Disposing?</param>
                protected virtual void Dispose(bool disposing)
                {
                        // close up our handles
                        if(disposing)
                                Disconnect();
                }
                #endregion
        }

        /// <summary>
        /// Represents errors that occur during the execution of the Wiimote library
        /// </summary>
        [Serializable]
        public class WiimoteException : ApplicationException
        {
                /// <summary>
                /// Default constructor
                /// </summary>
                public WiimoteException()
                {
                }

                /// <summary>
                /// Constructor
                /// </summary>
                /// <param name="message">Error message</param>
                public WiimoteException(string message) : base(message)
                {
                }

                /// <summary>
                /// Constructor
                /// </summary>
                /// <param name="message">Error message</param>
                /// <param name="innerException">Inner exception</param>
                public WiimoteException(string message, Exception innerException) : base(message,
innerException)
                {
                }

                /// <summary>
                /// Constructor
                /// </summary>
                /// <param name="info">Serialization info</param>
                /// <param name="context">Streaming context</param>
                protected WiimoteException(SerializationInfo info, StreamingContext context) :
base(info, context)
                {
                }
        }
}
```