UNIVERSITAT POMPEU FABRA

ESCOLA SUPERIOR POLITÈCNICA
ESTUDIS D'INFORMÀTICA

Projecte Fi de Carrera

Wii-immersion: Aprovechando las capacidades del Wiimote para la mejora de la sensación de inmersión

Iván Hernández Gutiérrez

Curs 2009-10

Director:

ENGINYERIA EN INFORMÀTICA

A Javi Agenjo por su apoyo,
su tiempo y su confianza.

A Dani por ser tan buen
compañero como amigo.

A mi familia por serlo.

A mi mejor amiga por todo.

A todos mis compañeros y amigos de la Universidad
por estar siempre online.

Y al aire acondicionado,
sin él este proyecto no hubiera sido posible.

Resumen

En este proyecto se proponen dos utilidades para mejorar la sensación de inmersión en un entorno virtual. Ambas utilidades pretenden aportar una interacción más intuitiva. Para ello, las dos se basan en capturar el movimiento del usuario a través de distintos dispositivos. En la primera utilidad se obtiene la posición de la cabeza del usuario y se visualiza la escena en base a dicha posición. Esto se consigue usando un Wiimote a modo de cámara infra-roja en combinación con unas gafas con leds infra-rojos para determinar la posición de la cabeza. La segunda utilidad consiste en un control gestual con seis grados de libertad. El control gestual permite manipular un objeto real, es decir, trasladarlo y rotarlo y que dicha manipulación se refleje en un objeto virtual. Manipulando el objeto es posible interactuar con el entorno virtual visualizado mediante el GTI Framework. La manipulación del control gestual permite interactuar de una forma muy natural, similar a como se interactuaría en un entorno real. Como objeto real se ha empleado un Wiimote en combinación con el periférico Motion Plus y una esfera luminosa ubicada en su extremo. El Motion Plus permite obtener la orientación, y mediante una cámara PlayStation Eye se detecta la esfera, que determina la posición del Wiimote.

Contenido

1. Introducción

La sensación de inmersión siempre ha sido un factor de gran importancia en las aplicaciones audiovisuales. Se podría decir que la inmersión se fundamenta en dos puntos clave. El primero es como se percibe el entorno virtual. Este primer punto se podría dividir en 5 aspectos, cada uno determinado por un sentido: vista, oído, gusto, tacto y olfato. Dado que nos ubicamos en el campo de las aplicaciones audiovisuales, tan solo disponemos normalmente de 2 de los 5 aspectos, el oído y la vista. El segundo punto en el que se fundamenta la sensación de inmersión es el de la interacción. Una interacción más natural con el entorno virtual nos proporciona una mayor sensación de inmersión. Es en este segundo punto, la interacción, donde centraremos el desarrollo del proyecto.

En este proyecto se abordará la implementación de dos utilidades que nos permitirán mejorar la experiencia inmersiva en aplicaciones 3D. Cada una de las utilidades representa una forma distinta de interactuar con la aplicación y las dos tienen en común que se fundamentan en actuar en base a los movimientos del usuario. La primera utilidad consiste en aplicar *view dependent rendering* mediante *head tracking.* Este método consiste en renderizar la escena en función de la posición del usuario con respecto a la cámara, normalmente ubicada encima o debajo de la pantalla. La segunda utilidad o forma de interacción con la aplicación es la que ha requerido la mayor parte del tiempo y esfuerzo de este proyecto y consiste en trasladar los 6 grados de libertad, traslación en los 3 ejes y orientación, de un objeto real a un objeto virtual. Es decir que si el objeto real se traslada o rota, el objeto virtual debe trasladarse y rotar del mismo modo. Esta forma de interacción se ha abordado tratando por separado la rotación del objeto y la traslación hasta que las dos han demostrado resultados admisibles, momento en que se han unido para formar la segunda utilidad.

Dado que el proyecto está formado por dos utilidades y cada una ha tenido su propio proceso de análisis, diseño e implementación, esto determinará la estructura del proyecto. En primer lugar nos centraremos exclusivamente en el análisis, diseño e implementación de la primera utilidad y después en el de la segunda. A su vez el análisis, diseño e implementación de la segunda aplicación también será dividido en dos, un capítulo relativo a obtener la orientación y otro referente a la posición.

La estructura del proyecto es poco ortodoxa ya que no se compone de una fase de análisis, una de diseño, una de implementación y otra de conclusiones. Dado que en este proyecto se han desarrollado dos utilidades y la segunda a su vez está compuesta de dos funcionalidades, cada una de estas partes requiere de su propio proceso de análisis, diseño, implementación y conclusiones. Redactar las etapas de las 3 funcionalidades de forma alternante hubiera resultado confuso ya que estaríamos saltando continuamente entre tres temas muy dispares. Por ello se decidió tratar de forma independiente las 3 funcionalidades manteniendo de este modo una coherencia temporal con el desarrollo del proyecto.

El proyecto se dividirá en 3 bloques o partes. La primera parte se centrará en la primera utilidad, el *view dependent rendering*. La segunda parte se centrará en la segunda

utilidad, el *motion controller*. Y la tercera parte unirá ambas utilidades en una única experiencia, a la que llamaremos *Wii-immersion*.

A continuación se hará un breve resumen del contenido de los capítulos de cada parte del proyecto:

- PARTE 1 - VIEW DEPENDENT RENDERING
  - **Capítulo 3**: Trata el análisis, diseño, implementación y conclusiones de la primera utilidad, el *view dependent rendering*.

- PARTE 2 - MOTION CONTROLLER
  - **Capítulo 4**: Es un breve capítulo de transición donde se describen las distintas utilidades que se consideraron previamente al *motion controller.*
  - **Capítulo 5**: Analiza el Wiimote como *motion controller*, valorando sus limitaciones y comparándolo con el PlayStation Move.
  - **Capítulo 6**: Trata el análisis, diseño, implementación y conclusiones de la primera función del *motion controller*, obtener la orientación.
  - **Capítulo 7**: Trata el análisis, diseño, implementación y conclusiones de la segunda función del *motion controller*, obtener la posición.
  - **Capítulo 8**: Se explica la integración de las dos funcionalidades desarrolladas en los capítulos 6 y 7, que conforman el *motion cotroller.*

- PARTE 3 - WII-IMMERSION
  - **Capítulo 9**: Se detalla la integración del *view dependent rendering* y el *motion controller* en una aplicación que visualiza un entorno virtual. También se relacionan los módulos implementados con la aplicación que genera y visualiza el entorno virtual.
  - **Capítulo 10**: Presenta las conclusiones finales y el trabajo futuro.
  - **Capítulo 11**: Glosario.
  - **Capítulo 12**: Índice de figuras.
  - **Capítulo 13**: Referencias.

Para la visualización del entorno virtual se ha utilizado el GTI Framework [1]. A continuación se detallan sus características y los entornos de pruebas creados con éste.

2. GTI Framework

El GTI Framework (Figura 1) es un conjunto de clases en C++ creadas con el objetivo de simplificar la creación de aplicaciones con gráficos tridimensionales. Está diseñada para ser independiente de la plataforma, el hardware, la estructura de particionamiento de la escena y las ecuaciones de renderizado. Las aplicaciones creadas con el GTI Framework son aceleradas por hardware, pueden ser ejecutadas de forma clusterizada y su *pipeline* es totalmente programable.

2.1 Características

A continuación se detallarán algunas de estas características:

- **Independencia del sistema operativo**: las aplicaciones creadas con el GTI Framework pueden ejecutarse tanto en Windows como en sistemas Unix. Esto se consigue mediante el uso de librerías externas, como SDL (*Simple DirectMedia Layer)* o GLUT (OpenGL Utility Library Toolkit).

- **Multiplataforma**: las aplicaciones desarrolladas mediante el GTI Framework son compatibles tanto con los formatos *big-endian* como con *little-endian.* Los importadores de archivos binarios resuelven la conversión. De este modo podemos importar archivos DDS (Direct Draw Surface) y Targa (TGA).

- **Aceleración por hardware**: las aplicaciones creadas con el GTI Framework se benefician del hardware destinado a gráficos del sistema. La API de gráficos programable empleada es OpenGL 2.0. Las extensiones son extraídas mediante GLEW (OpenGL Extension Wrangler Library). El *framework* está diseñado para trabajar con las técnicas más rápidas (*vertex buffer objects, geometry, vertex and pixel shaders, render to texture...)* a fin de conseguir el mejor rendimiento visual. Dado que el *framework* no emplea extensiones hardware su compatibilidad es muy alta.

- ***Pipeline* completamente programable**: en las aplicaciones desarrolladas mediante el GTI Framework podemos definir libremente las ecuaciones de renderizado de las mallas de triángulos. No existen restricciones en cuanto a las pasadas de renderizado y no se emplea *fixed pipeline* para garantizar la portabilidad. La definición del renderizado se consigue mediante archivos de texto, sin necesidad de recompilar el código fuente.

- **Fácil integración con aplicaciones de modelado 3D**: el *framework* integra un importador de escenas en formato FBX. El formato FBX es un formato estándar soportado por la mayoría de aplicaciones de modelado 3D. De este modo es muy fácil exportar mallas, animaciones materiales o escenas completas a nuestra aplicación.

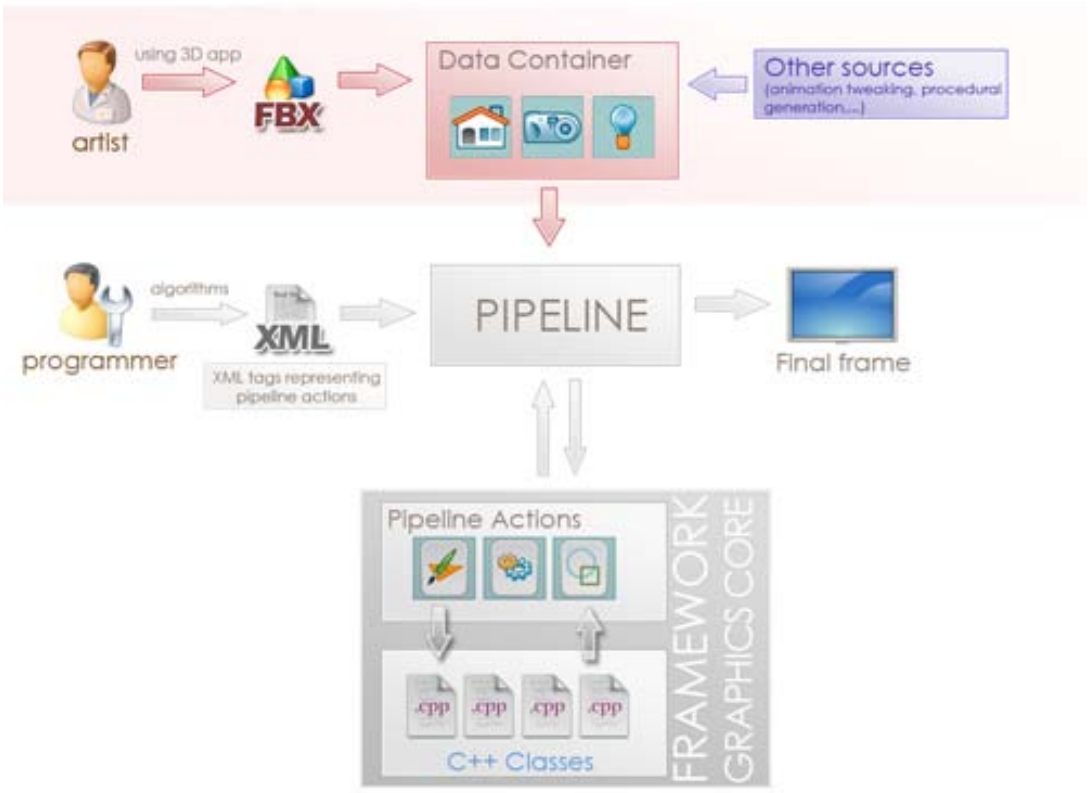

Figura 1 Diseño del funcionamiento del GTI Framework

- **Soporte para renderizado estereoscópico:** las aplicaciones programadas con el GTI Framework pueden funcionar de forma estereoscópica con muy pocos cambios y sin tener que subministrar una *pipeline* entera con soporte para estereoscopía.

- **Gran cantidad de componentes y utilidades gráficas incluidas:** además de soporte para texturas comprimidas, *Shaders, Render to Texture, Vertex Buffers,* el *framework* incorpora varias clases diseñadas para gestionar cámaras, *frustums,* mallas, etc.

- **Transparencia en entornos de clusterizado:** gracias al sistema de sincronización se puede tener varias variables compartiendo exactamente el mismo valor en distintas aplicaciones.

2.2 Entorno de pruebas

Para testear la primera utilidad se creó una Cornell Box con varios objetos en ella a distintas alturas y distancias (Figura 2). Para la segunda aplicación se empleó de nuevo una Cornell Box como escena pero con un modelo 3D de un Wiimote en su interior (Figura 3).

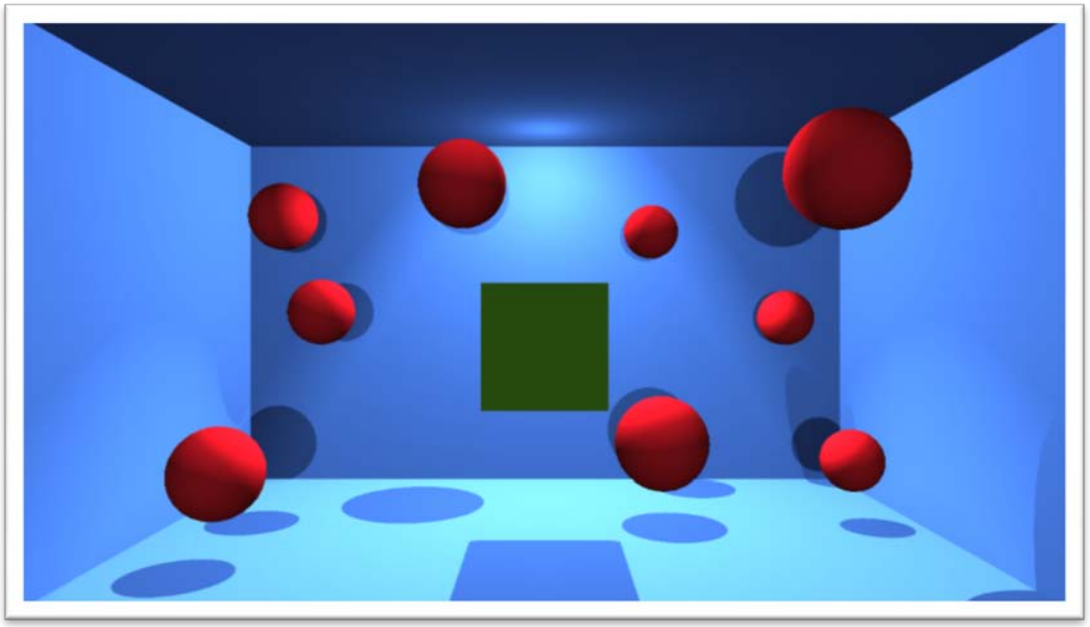

Figura 2 Cornell Box de la utilidad 1, v*iew dependent rendering*

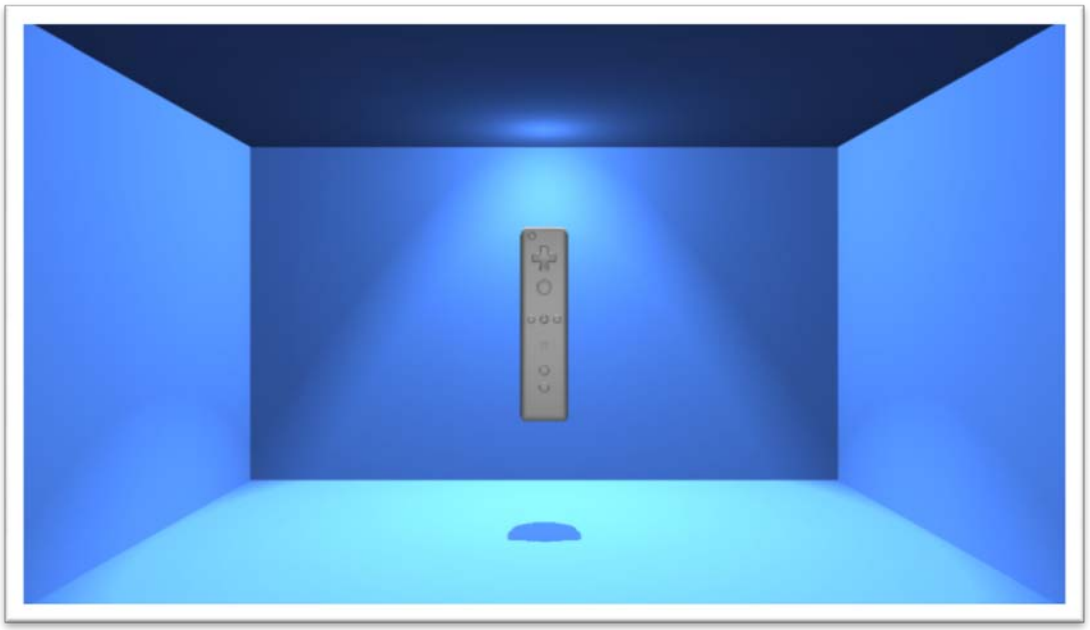

Figura 3 Cornell Box de la utilidad 2, *motion controller*

PARTE 1 - VIEW DEPENDENT RENDERING

3. View dependent rendering mediante head tracking

3.1 Análisis

Se denomina *head tracking* a la técnica que permite detectar y seguir la posición de la cabeza del usuario mediante una cámara. A veces se emplean los términos *head tracking* y *face tracking* como sinónimos. Sin embargo el termino *face tracking* hace referencia a un proceso más complejo que el *head tracking* ya que no tan solo se detecta la cabeza del usuario, sino también los ojos, nariz y boca de éste.

El *head tracking* tiene varias utilidades. En el mundo de la fotografía, por ejemplo, las cámaras que disponen de software para realizar *head tracking* permiten postprocesar con mayor detalle las zonas detectadas por el software. También puede emplearse como detector de presencia para encender o apagar una webcam, etc.

Para este proyecto se emplea el *head tracking* para realizar *view dependent rendering*. Esto consiste en renderizar una escena en función de la posición de la cabeza del usuario. El *view dependent rendering* hace años que se emplea. Ejemplos de ello son los dispositivos TrackIR [2] o la recreativa Police 24/7, sin embargo no ha sido hasta la utilidad creada por Johnny Chung Lee, *Head Tracking for Desktop VR Displays using the Wii Remote* [3], que el view dependent rendering ha empezado a generar interés.

A continuación analizaremos las distintas opciones disponibles para realizar *view dependent rendering* mediante *head tracking*. En primer lugar nos centraremos en los métodos actuales para realizar *head tracking*, destacando sus inconvenientes y ventajas. Tras explorar los distintos métodos diferenciaremos entre las posibles formas de renderizar la escena mediante los datos que nos aporta el *head tracker.* Y finalmente estudiaremos las tecnologías disponibles para realizar head tracking.

3.1.1. Head tracking: metodologías

Los métodos para realizar *head tracking* se podrían clasificar en dos variantes en función del hardware empleado y en otras tres variantes en función de los grados de libertad que se deseen obtener. En primer lugar analizaremos las dos variantes hardware.

Por un lado tenemos los que emplean cámaras convencionales (webcams y similares) y detectan en cada *frame* de video la cabeza del usuario mediante algoritmos de procesamiento de imagen. Esta metodología presenta varios inconvenientes, en primer lugar procesar cada frame para detectar la cabeza del usuario requiere un tiempo considerable, por ejemplo la implementación que incluye el SDK de OpenCV [4] tarda una media de 300 ms en un ordenador de gama alta, equivalente a unos 3 *fps* (imágenes por segundo). Para realizar *head tracking* en tiempo real deberíamos procesar cada *frame* en un tiempo máximo de 16,6 ms, considerando 60 *frames* por segundo suficiente para procesamiento en tiempo real. Pero

aunque consiguiéramos rebajar el tiempo a esas cotas, todavía deberíamos afrontar otro problema más complejo, la imprecisión. Los *head trackers* por procesamiento de imagen tan solo nos dan una noción aproximada de donde está ubicada la cabeza del usuario. Además la aproximación suele tener mucho ruido, apreciándose variaciones constantes en la estimación de la distancia de la cabeza, la cual es estimada en base al tamaño del área reconocida como cabeza.

La segunda variante hardware se caracteriza por el uso de cámaras infra-rojas. Estas cámaras tan solo capturan luz infra-roja (la cual es invisible para el ojo humano) y nos proporcionan la posición en el plano de la imagen de las fuentes de luz infra-roja captadas. La idea de esta variante consiste en emplear luces infrarrojas, que en este trabajo serán referidas como leds IR a modo de marcadores. Cuantos más marcadores usemos más grados de libertad conseguiremos. En cuanto al tiempo de procesamiento de cada *frame* es ínfimo, ya que la detección se realiza por hardware. La precisión tampoco es un problema, obteniendo la posición en el plano de los distintos marcadores sin ruido notable y con una desviación estándar de pocos milímetros en distancias cercanas y escasos centímetros en distancias más considerables, dependiendo de la resolución de la cámara y del ángulo de visión de ésta. Y la última ventaja con respecto a la primera variante es su correcto funcionamiento independientemente de las condiciones de iluminación, ya que para realizar el *tracking* con el primer sistema se requieren unas condiciones de iluminación óptimas.

En cuanto a las variantes del *head tracking* en base a los grados de libertad diferenciaremos 3 tipos: con 2 grados de libertad, con 4 grados de libertad y con 6 grados de libertad.

En la variante con 2 grados de libertad solo tomamos en consideración la posición de la cabeza en el plano imagen, es decir, no tenemos en cuenta la distancia a la cámara. Esta variante presenta la peculiaridad de que una misma traslación del usuario es interpretada de distinta forma en función de la distancia de éste a la cámara. Es decir, que si el usuario realiza una traslación de su cabeza cerca de la cámara la distancia recorrida en píxeles será mucho mayor que si realiza esa misma traslación a más distancia de la cámara. Relacionándolo con las variantes hardware, para conseguir dos grados de libertad con la primera variante, la de procesamiento de la imagen, tan solo debemos preocuparnos de calcular la posición de la cabeza en el plano que vendrá dada por el centro del área determinada como cabeza. Y en la variante con cámara infra-roja tan solo será necesario usar un único led IR como marcador.

En la variante con 4 grados de libertad consideramos las traslación de la cabeza en los 3 ejes de coordenadas, es decir, añadimos un nuevo grado de libertad determinado por la distancia de la cabeza a la pantalla y un segundo grado más de libertad ya que podemos calcular el *roll,* alabeo en castellano, de la cabeza mediante el ángulo que forma el segmento formado por los dos marcadores con la horizontal . Esta variante, además de añadir un grado de libertad, subsana la peculiaridad de la variante anterior. Simplemente multiplicando la distancia en pixeles recorrida en el plano *XY* por la distancia estimada en *z* conseguimos que por ejemplo una traslación de la cabeza 1 metro sea interpretada igual independientemente de la distancia a la pantalla. Relacionándolo con las variantes hardware, en la variante por procesamiento de imagen se obtiene la distancia en función del tamaño del área reconocida

como cabeza. Y en la variante con marcadores se emplean 2 marcadores colocados uno en cada extremo de la cabeza. De este modo se estima la distancia en base a los píxeles de distancia entre los dos marcadores, es decir, que si los marcadores están más distanciados indicará que estamos cerca de la cámara y si por el contrario los marcadores están a pocos píxeles de distancia indicará que estamos muy alejados de la cámara.

Finalmente tenemos la variante con 6 grados de libertad. Aquí además de considerar la traslación en los 3 ejes también se considera el ángulo de la cabeza con respecto a los 3 ejes. Esta variante es la más compleja, pero permite reflejar todos los movimientos de la cabeza. Relacionándolo una vez más con las variantes hardware, en la variante por procesamiento de imagen se deben emplear algoritmos para obtener la correspondencia entre los distintos píxeles que conforman la cara entre un *frame* y el siguiente. Para obtener dicha correspondencia se emplean métodos como Lucas Kanade [5]. Una vez detectada la correspondencia entre algunos puntos se genera una maya triangular y se interpreta la rotación en base a la variación de las normales de los polígonos de la maya triangular. En la variante con marcadores leds IR es mucho más sencillo obtener los 6 grados de libertad, simplemente se deben emplear 3 marcadores posicionados de tal manera que dibujen un triángulo y obtener la orientación gracias a la normal de éste. De esta forma es cómo funcionan los dispositivos TrackIR anteriormente citados.

3.1.2. View dependent rendering: métodos de renderizado

En este apartado se analizaran las distintas maneras de usar los datos obtenidos por el *head tracker* para obtener *view dependent rendering.* Los dos métodos actúan sobre la cámara que muestra la escena renderizada.

El primero es el más sencillo y consiste en reproducir los movimientos de la cabeza del usuario mediante la cámara, es decir que si la cabeza del usuario se traslada o rota (si disponemos de 6 grados de libertad) la cámara reproduzca el mismo movimiento. De este modo podemos interactuar en distintos entornos realizando movimientos con nuestra cabeza. Sin embargo los movimientos se ven limitados por el tamaño del *display* y el rango de visión de la cámara, lo que a la práctica se traduce en movimientos de "asomo", que en inglés se denomina *lean movements*, movimientos breves como asomarse, inclinarse o ladear ligeramente la cámara, que pueden ser escalados para ser más notorios, pero hasta cierto grado, ya que a mayor escalado más difícil se hace el control de la cámara. Este método es el empleado normalmente en los dispositivo TrackIR.

El segundo método a pesar de ser más complejo, solo necesita dos marcadores, es decir, conocer únicamente la posición de la cabeza. En éste los parámetros obtenidos por el *head tracker* son mapeados para simular una propiedad de la vista denominada *parallax* o paralaje en castellano. El *parallax* es la desviación angular de la posición aparente de un objeto dependiendo del punto de vista elegido (Figura 4). El efecto que se consigue al desplazar la cabeza con éste método es similar a mirar a través de una ventana. De este modo convertimos nuestro *display* en una ventana al mundo virtual. Este método es el empleado en la aplicación

Head Tracking for Desktop VR Displays using the Wii Remote de Johnny Chung Lee y es el que ha despertado de nuevo el interés en el *view dependent rendering*.

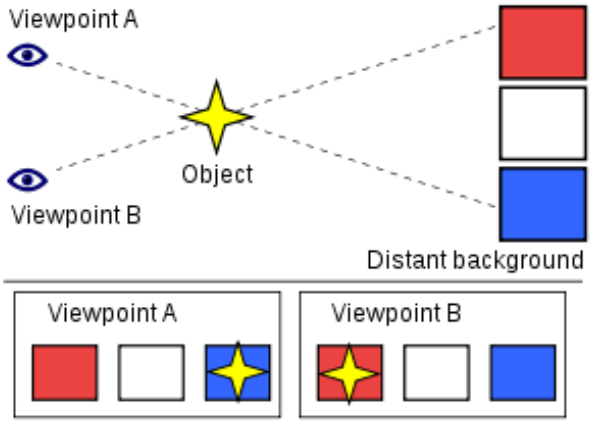

Figura 4 Ejemplo de *parallax*

3.1.3. Tecnologías: estudio de prestaciones

En este punto del análisis compararemos las distintas tecnologías disponibles para realizar *head tracking*. En concreto analizaremos 4 posibles opciones: las webcams domésticas, el PlayStation Eye, los sistemas TrackIR y el Wiimote.

En primer lugar analizaremos las webcams domésticas. Por domésticas nos referimos a que están pensadas para el uso doméstico y eso implica un diseño enfocado a tareas como tomar fotografías o realizar videoconferencias. Para la tarea fotográfica las webcams domésticas escalan en resolución, siendo hoy en día habituales cámaras de 1.3 Megapíxeles y de resoluciones superiores. Para la tarea de la videoconferencia resulta vital no enviar un gran número de imágenes por segundo ya que necesitarían consumir más ancho de banda, además una videoconferencia se caracteriza por ser una interacción muy estática. Es por ello que el número de imágenes por segundo habitual en cámaras domésticas no suele ser superior de 30.

El PlayStation Eye es la cámara diseñada para la PlayStation3 y sucesora del EyeToy, la cámara de la PlayStation2. Gracias a drivers no oficiales es posible usar la cámara en sistemas Unix y Windows. Sus prestaciones son destacables. Permite capturar imágenes a 640x480 píxeles a 60 *fps* y a 320x240 a 120 *fps*. Estas prestaciones la hacen un dispositivo ideal para realizar *tracking* en tiempo real. Sin embargo, al igual que las webcams domésticas requiere de técnicas de procesamiento de imagen para realizar la detección.

Los sistemas TrackIR fueron puestos a la venta en 2001 por NaturalPoint Inc y permiten controlar la cámara mediante el movimiento de la cabeza con 6 grados de libertad en los juegos para Windows con soporte para TrackIR implementado. Actualmente existen 6 versiones de TrackIR disponibles en el mercado (Figura 5). Todos ellos pertenecen a la variante hardware que emplea una cámara infra-roja y marcadores y se diferencian por el número de

puntos que pueden detectar, la resolución, el ángulo de visión y el número de imágenes por segundo que son capaces de procesar. Los marcadores pueden ser leds IR montados sobre un dispositivo con alimentación o bien reflectores que reflejan la luz de los leds IR montados sobre la cámara.

TrackIR model comparison

Model	Sensor resolution	FPS	Angle	Subpixel precision	DOF	Released	Latest software
TrackIR 1	60k pixels (e.g. 300x200)	60	33		2	2001	Version 3.x
TrackIR 2	60k pixels (e.g. 300x200)	100	33		2	2003	Version 3.x
TrackIR 3	355x288	80	33		2 or 6	2004	Version 4.x
TrackIR 3:PRO	355x288	120	33		2 or 6	2004	Version 4.x
TrackIR 4:PRO	355x288 (sub-sampled at 710x288)	120	46	1/20th	6	2005	Version 4.x
TrackIR 5	640x480	120	51.7	1/150th	6	2009	Version 5.x

Figura 5 Tabla comparativa de los distintos modelos de TrackIR

Por último se analizaron las prestaciones del Wiimote como dispositivo de *tracking.* El Wiimote es el controlador empleado en la consola de videojuegos Nintendo Wii. Más adelante entraremos a detallar todas las características del Wiimote. En este apartado únicamente analizaremos la cámara que éste tiene instalada y que emplea para detectar los leds IR posicionados en la erróneamente llamada *sensor bar*, de la consola.

Sus características son las siguientes:

- Sensor monocromático que únicamente capta luz infra-roja.
- Detecta hasta un máximo de 4 fuentes de luz infra-roja por hardware.
- Resolución de 128x96 píxeles que submuestreada se convierte en 1024x768.
- Ángulo de visión de 45 grados.
- 100 imágenes por segundo. Realmente no se transmiten 100 imágenes sino que se envían las coordenadas *XY* de los puntos detectados una media de 100 veces por cada segundo mediante bluetooth.

3.1.4. Resumen del análisis

Repasemos los datos obtenidos en el análisis:

- Métodos para realizar *head tracking*:
  - Según el hardware:
    - Cámaras RGB
    - Cámaras infra-rojas
  - Según los grados de libertad:
    - 2 grados
    - 4 grados
    - 6 grados

- Métodos de renderizado:
  - La cámara reproduce los movimientos de la cabeza
  - *Parallax* (efecto ventana)

- Tecnologías:
  - Cámaras RGB:
    - Cámaras web
    - PlayStation Eye
  - Cámaras infra-rojas:
    - TrackIR
    - Wiimote

3.2 Diseño

Una vez analizados las distintas opciones disponibles a nivel de hardware, software y de funcionalidades. Se explicaran las decisiones tomadas gracias al proceso de análisis, es decir que tecnología o tecnologías han sido elegidas y de qué forma realizar el *head tracking* y *view dependent rendering*.

3.2.1 Conclusiones del análisis

El proceso de selección se inició descartando tecnologías ya que éstas son condicionantes de los posibles métodos para realizar *head tracking*. Dado que únicamente se disponía de una cámara web y un Wiimote descartamos las otras dos opciones. Analizando las prestaciones de la cámara web y del Wiimote sopesamos los pros y los contra de cada uno. La principal ventaja de la cámara web es que permite programar una utilidad libre de marcadores sin embargo sería una utilidad que para obtener niveles de precisión similares a una misma utilidad con marcadores requería de múltiples optimizaciones. Pero el principal inconveniente de la cámara web y el que decantó la balanza es su baja tasa de imágenes por segundo, 30 *fps* son muy pocos como para obtener un *tracking* en tiempo real suave y preciso. El proceso de análisis hizo decidirse por el Wiimote, dado que su único inconveniente era la necesidad de usar marcadores.

La siguiente decisión consistió en determinar el número de marcadores a usar, dado que esto condicionaría los grados de libertad y el método de renderizado. Se idearon dos formas de sostener los marcadores en la cabeza del usuario, mediante una gorra y mediante unas gafas. Finalmente las gafas fue la opción elegida. El motivo fue sencillamente porque es posible adquirir gafas que se emplean en lampistería (Figura 6) que incorporan dos leds de luz blanca a modo de linterna y reemplazar dichos leds por leds IR.

Figura 6 Gafas con leds incorporados

Al emplear las gafas limitamos los marcadores a 2 y por lo tanto los grados de libertad a 4, la traslación en los 3 ejes y el *roll*. Usar las gafas también condicionó el método de renderizado a elegir, se descartó el primer método ya que al no disponer de la orientación completa ofrecía un control limitado, siendo finalmente el método *parallax* el elegido.

En los siguientes apartados se explicará cómo obtener la posición de la cabeza mediante el Wiimote y detallaremos en qué consiste el método parallax, modelando de este modo los distintos bloques de los que se compondrá la utilidad.

3.2.2 Obtención de la posición de la cabeza mediante el Wiimote

Independientemente de la librería empleada el Wiimote puede detectar hasta 4 marcadores infra-rojos y estos marcadores son mapeados en una imagen de 1024x768 píxeles. En el apartado de implementación se detallará como obtener los datos de la cámara del Wiimote, aquí únicamente explicaremos como usaremos dichos datos.

En este proyecto se emplearon dos marcadores instalados en unas gafas para determinar la posición de la cabeza del usuario. Para calcular dicha posición se emplearon los datos obtenidos por la cámara que fueron obtenidos como se explica a continuación

Se comprueba si los dos marcadores son visibles. En caso afirmativo se procede a estimar la posición de la cabeza. En primer lugar se debe determinar la distancia a la pantalla/cámara, es decir, la magnitud del segmento perpendicular a la cámara que empieza en ésta y finaliza en nuestra cabeza. Para ello debemos conocer previamente tres datos:

- Los radianes por pixel
  - Viene dado por la relación entre el ángulo de visión de la cámara y la resolución horizontal de ésta, es decir π/4 radianes (45 grados) entre 1024 píxeles.

- La distancia en milímetros entre los dos marcadores
  - Se obtiene simplemente midiendo dicha distancia.

- La distancia en píxeles entre los dos marcadores.
  - La debemos calcular nosotros dada la posición de los dos marcadores en rejilla de 1024x768 que define la imagen de la cámara.

Conocida la distancia en píxeles entre los dos marcadores y los radianes por píxel podemos deducir el ángulo que se ilustra en la figura siguiente (Figura 7):

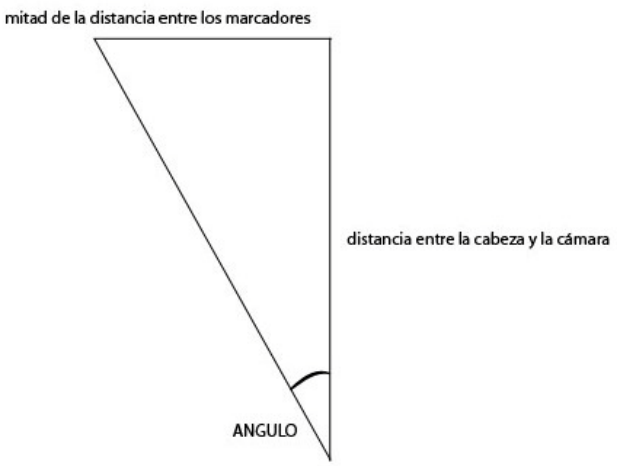

Figura 7 Relación entre la distancia de los marcadores y la distancia de la cabeza con la cámara

El ángulo se obtiene multiplicando los radianes por píxel por la mitad de la distancia entre los marcadores en píxeles. Una vez obtenido el ángulo podemos emplear la tangente de éste para determinar la distancia de la cabeza. Sabemos que la tangente de un ángulo es igual a dividir la magnitud del cateto opuesto entre la magnitud del cateto contiguo, que en este caso resulta ser la distancia que buscamos. De este modo se puede determinar que la distancia que buscamos puede ser calculada dividiendo la magnitud del cateto opuesto tomando el valor real en milímetros y no en píxeles y dividiendo este por la tangente del ángulo encontrado.

$$\text{tangente}(\text{angulo}) = \frac{\text{mitad de la distancia entre los marcadores en milímetros}}{\text{distancia entre la cabeza y la cámara en milímetros}}$$

Para calcular la posición en X e Y de la cabeza debemos calcular el punto medio en píxeles de los dos marcadores. Sabiendo el punto medio, los radianes por píxel , la resolución de la cámara y la recién obtenida distancia entre la cabeza y la cámara, podemos calcular el desplazamiento respecto al centro del plano imagen tal que así:

*X de la cabeza = seno (radianes por píxel * (punto medio en X - 512)) * Z de la cabeza*

*Y de la cabeza = seno (radianes por píxel * (punto medio en Y - 384)) * Z de la cabeza*

Donde 512 y 384 son la mitad de las resoluciones horizontal y vertical de la cámara respectivamente y Z de la cabeza es la distancia entre la cabeza y la cámara calculada en la anterior expresión.

3.2.3 Parallax view dependent rendering

Una vez conocida la posición de la cabeza, gracias a nuestro *head tracker,* dispondremos de los datos necesarios para realizar el *view dependent rendering.* A continuación se detallará en qué consiste esta forma de *view dependent rendering* a la que nos referiremos a partir de ahora como *parallax view dependent rendering*.

El parallax view dependent rendering está compuesto de dos acciones. La primera consiste en trasladar la cámara de igual manera que ser haría en un *view dependent rendering* normal. La segunda acción consiste en modificar la perspectiva mediante un *frustum*. Un *frustum* es una pirámide truncada y normalmente se define mediante 6 parámetros, l*eft, right, top y bottom,* que delimitan la parte rectangular del *frustum* y *near* y *far* que delimitan los extremos del *frustum* (Figura 8).

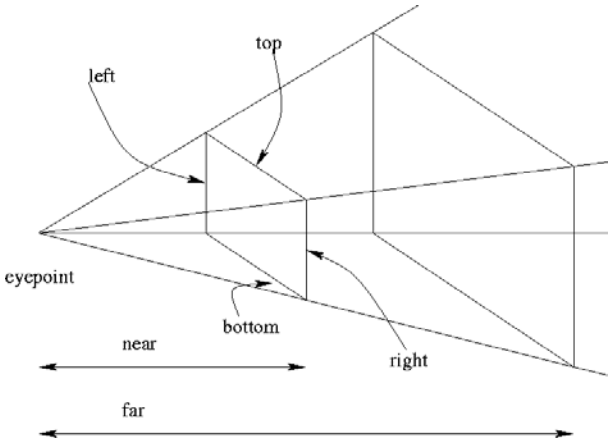

Figura 8 Representación de un *frustum*

La clave del *parallax view dependent rendering* consiste en cambiar el *frustum* de tal forma que este siga conteniendo un mismo marco estático en éste (Figura 9). Nos referiremos

con el término marco al elemento invisible a través del cual se visualizará el efecto que queremos conseguir. Es decir el marco es la ventana en el mundo real. En nuestro mundo virtual deberemos ajustar la cámara al marco para conseguir el *parallax view dependent rendering*.

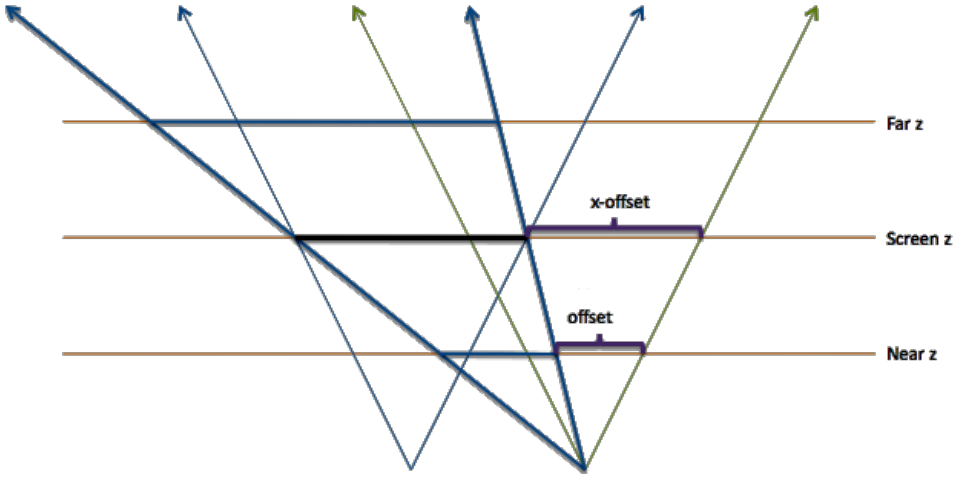

Figura 9 Explicación del *parallax view dependent rendering*

En la figura superior se muestra de forma ejemplificada cómo funciona el *parallax view dependent rendering*. Las parejas de flechas del mismo color representan *frustums* vistos desde arriba y la línea negra es nuestro marco El objetivo es que al desplazar la cámara el marco siga dentro del *frustum*. El *frustum* azul claro (izquierda) representa el estado inicial. A continuación supongamos que se traslada la cámara en X hacia la derecha, haciendo esto obtendríamos el frustum verde (derecha) si lo calculásemos como un *view dependent rendering* normal. En este caso no queremos obtener el verde, sino el azul oscuro (central), ya que éste seguirá manteniendo el marco dentro de él. Para ello simplemente debemos restar al *left* y al *right* el *offset* en *X* y al *top y* al *bottom* el *offset* en *Y*. Dado que hemos dicho que el *frustum* determina la proyección, obsérvese que el *frustum* azul oscuro distorsiona la perspectiva con respecto al *frustum* original, obteniendo de este modo el efecto visual deseado.

3.2.4 Bloques del diseño

Una vez planteado el funcionamiento de la utilidad podemos dividir en distintos bloques funcionales los procesos que se deben realizar. La utilidad se dividirá en 3 bloques. El primero se encargará de realizar la conexión y desconexión del Wiimote y obtener los datos que nos aporte su cámara. El segundo bloque recibirá los datos de la cámara y los convertirá en la posición de la cabeza. Y finalmente el último bloque recibirá los datos de la posición de la cabeza y los empleará para modificar la posición y proyección de la cámara (Figura 10).

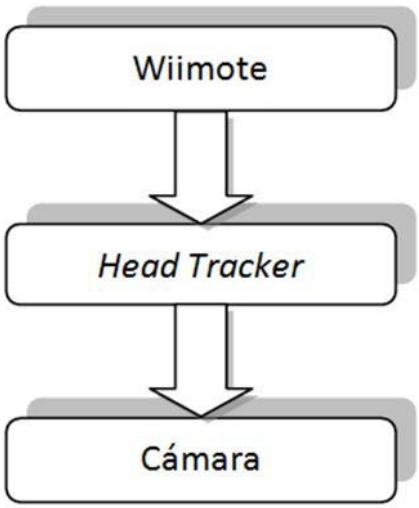

Figura 10 Bloques funcionales del *view dependent rendering* mediante *head tracking*

3.3 Implementación

En este apartado se detallarán como han sido implementados los distintos módulos y sus funcionalidades. También se presentará la librería utilizada para obtener los datos del Wiimote.

3.3.1 Librerías para gestionar el Wiimote

Para implementar el módulo para obtener los datos del Wiimote se estudiaron distintas librerías. De entrada se descartaron varias librerías: *moteJ*, *WiiRemoteJ* y *WiiuseJ* por estar escritas en *Java*. También se descartaron *CWiid* y *WiiYourself*! [6] por ser exclusivas para un sistema operativo Linux y Windows respectivamente. WiimoteLib también fue descartada por estar escrita en C#. Finalmente la elegida fue *wiiuse*. Aunque no tenía tantas funciones como otras librerías, la ventaja de *wiiuse*, implementada en C, era la posibilidad de usarla tanto en Windows como en Linux, haciendo fácil la portabilidad.

Algunas de las funcionalidades del *wiiuse* son:

- Soporte para acelerómetros.
- Infrarrojos.
- Soporte de algunas extensiones (*Nunchuck* y *Classic Controller*).

Dado que nos permitía fácil portabilidad y obtener los datos de la cámara infra-roja *wiiuse* fue la librería elegida en primer lugar y la que se empleó para implementar el *head tracker.* Sin embargo al iniciar el análisis de la segunda utilidad se tuvo que cambiar de librería.

El motivo fue sencillamente porque *wiiuse* no tiene soporte para la extensión *Wii Motion Plus*, totalmente imprescindible para implementar la segunda utilidad.

WiiYourself! fue la librería elegida finalmente. Sin embargo se tuvo que sacrificar la compatibilidad con sistemas *Linux*.

La lista de funcionalidades soportada por la versión 1.15 de *WiiYourself!* es muy extensa:

- Soporte para varios Wiimotes.
- Soporte para Nunchuck, Classic Controller, **Wii Motion Plus** y Wii Balance Board.
- Lectura de: batería, botones, acelerómetros e infra-rojos.
- Estimación de la orientación.
- Permite establecer los LEDS y el *rumble* del mando.
- Conexión bluetooth con cualquier *stack.*

Para no usar dos librerías para gestionar los Wiimotes, la primera utilidad fue re-implementada usando *WiiYourself*! 1.15.

3.3.2 Módulos y funcionalidades

Como vimos en el punto 3.2.4 la utilidad se compone de tres bloques principales. El primero debe encargarse de gestionar el Wiimote. Esto implica realizar la conexión y desconexión del Wiimote y obtener los datos que nos aporte su cámara. El segundo bloque debe recibir los datos de la cámara y convertirlos en la posición de la cabeza. Y finalmente el último bloque debe usar los datos de la posición de la cabeza para modificar la posición y proyección de la cámara de la aplicación.

Para realizar la conexión con el Wiimote y obtener las posiciones de los leds IR captados por la cámara se ha implementado la clase **WiimoteManager**. Esta clase actúa a modo de interfaz de la librería WiiYourself! y tiene como peculiaridad que las distintas instancias de ésta se ejecutan en hilos independientes. Es decir, cada instancia está ideada para gestionar un Wiimote. La clase se encarga de realizar la conexión y desconexión del Wiimote y de solicitar los datos del nuevo estado del Wiimote. Una vez realizada la conexión, la clase ejecutará el hilo donde actualizará el estado del Wiimote con una frecuencia media de 100Hz. De este modo tendremos siempre actualizada la información de la posición de los leds IR captados por la cámara.

Para obtener la posición de la cabeza gracias a los datos del Wiimote se ha implementado la clase **WiimoteHeadTracker**. Ésta tiene básicamente dos funciones. La primera es inicializar todos los parámetros necesarios para convertir las posiciones de los leds IR en la posición de la cabeza. Entre los parámetros que se inicializan hay cinco de gran importancia. El primer parámetro es la distancia en milímetros entre los dos leds IR de nuestras gafas. Dar un valor preciso a este parámetros nos permite obtener mejores

mediciones de la posición de la cabeza con respecto a la cámara. Los otros cuatro parámetros de gran importancia son cuatro matrices. En ellas se guardarán las matrices de vista y proyección de la cámara. Utilizar estas cuatro matrices hace posible activar y desactivar el modo de renderizado mediante el *head tracker.* La segunda función que implementa la clase se encarga de obtener los datos de la posición de los leds IR y convertirlos en las coordenadas de la cabeza con respecto a la cámara.

Para visualizar la escena en función de la posición de la cabeza del usuario, se ha creado un método en la clase **Camera** del GTI Framework que transforma las matrices de la vista y la proyección de la cámara utilizando los datos de la posición de la cabeza proporcionados por el *head tracker.*

Además de activar y desactivar el *head tracker* también podemos gestionar algunos parámetros de éste. Para ello se hará uso del teclado o de un segundo Wiimote. Dado que la cámara puede ser colocada a cualquier altura, es necesario que el usuario indique cuando está preparado para iniciar el uso del *head tracker.* Al indicarlo, la visualización 3D compensará la componentes *Y* para centrar la vista. También es posible posicionar la cámara de tal modo que encaje con el marco donde el efecto *parallax* es más notable. De forma opcional, el usuario puede acentuar o rebajar la profundidad de la escena sin necesidad de alejarse o acercarse a ésta. Finalmente, se permite incrementar y decrementar el factor de escalado en *X* e *Y*. Incrementar el factor de escalado permite simular un mayor desplazamiento.

3.4 Conclusiones

Una vez implementada la utilidad se evaluó su funcionamiento para detectar posibles inconveniencias, limitaciones y aspectos a mejorar.

A continuación se muestran algunas capturas que ejemplifican su funcionamiento.

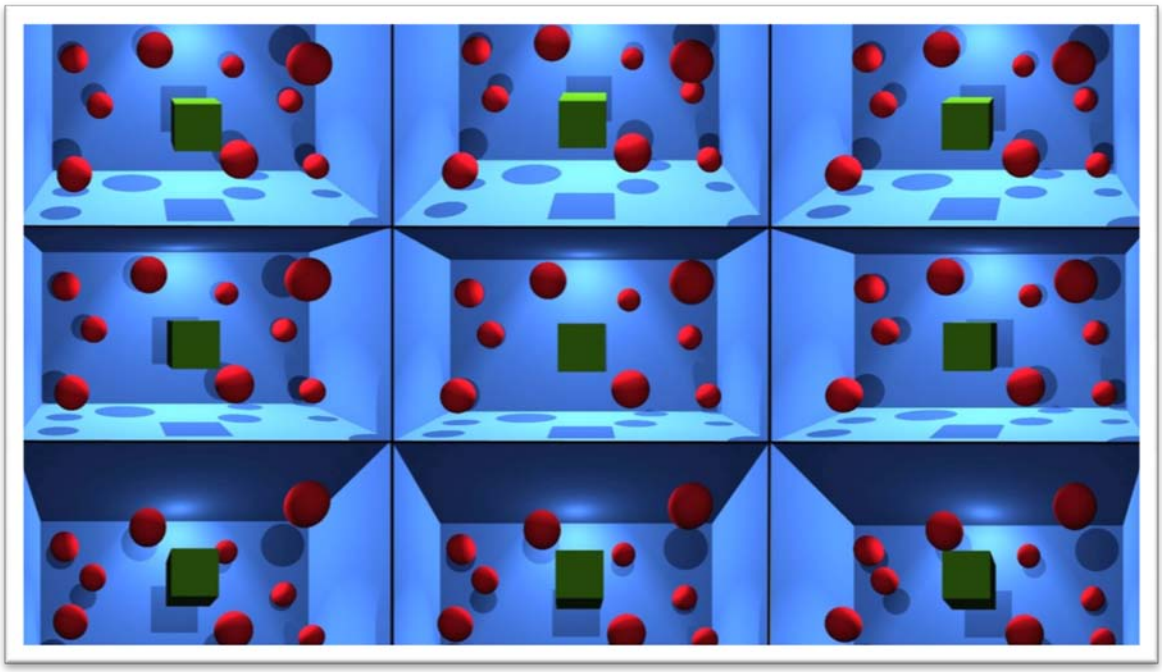

Figura 11 Escena visualizdas en función de distintas posiciones de la cabeza del usuario.

En las 9 capturas superiores (Figura 11) se puede ver como se visualiza la escena en función de la posición de la cabeza del usuario. La imagen central representa como se vería la escena posicionando la cabeza justo en el centro de la cámara. La imagen de su izquierda es el renderizado de la escena si desplazamos la cabeza hacia la izquierda, etc.

A continuación se muestra una secuencia de fotografías (Figuras 12, 13, 14, 15, 16 y 17) en las que se puede ver cómo responde la visualización a la posición de la cabeza del usuario.

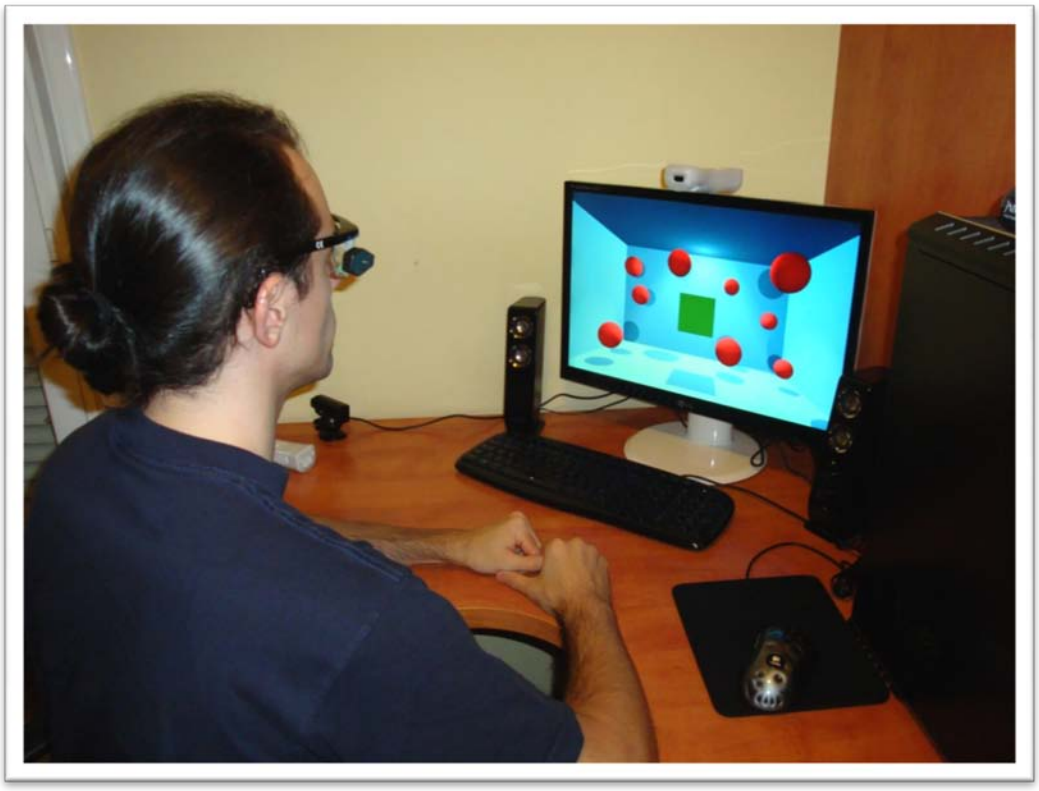

Figura 12 View dependent rendering con usuario centrado

Figura 13 View dependent rendering con usuario cerca de la cámara

Figura 14 View dependent rendering con usuario a la izquierda de la cámara

Figura 15 View dependent rendering con usuario a la derecha de la cámara

Figura 16 View dependent rendering con usuario por encima de la cámara

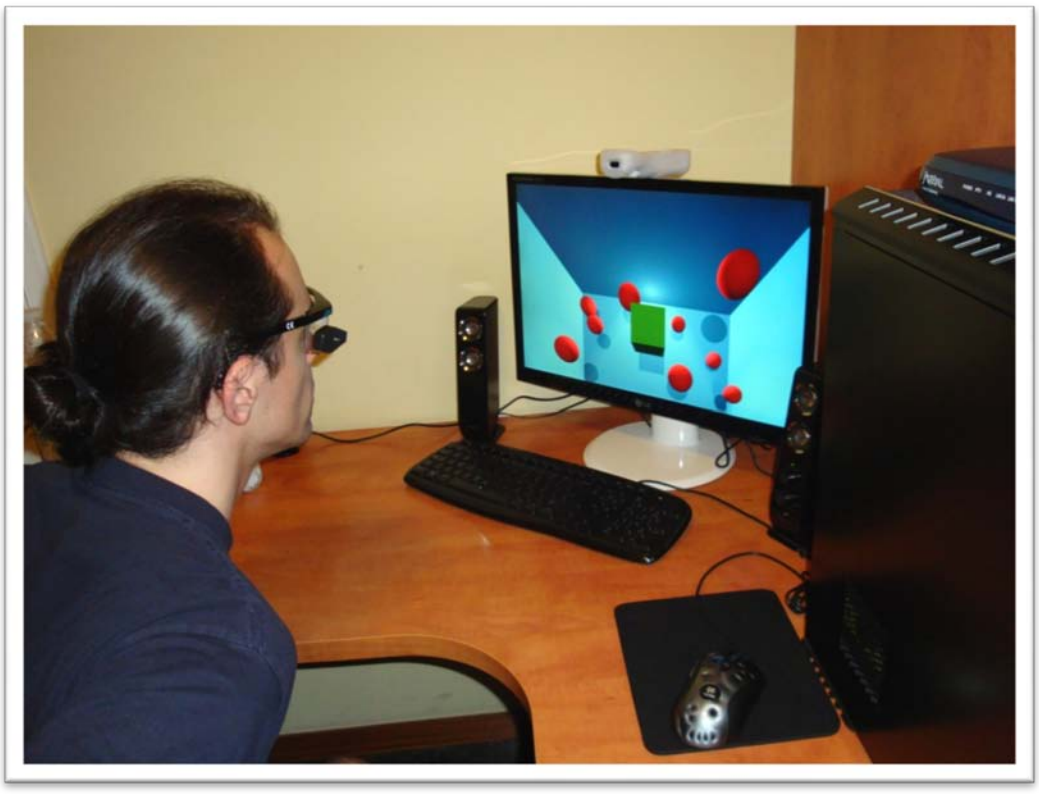

Figura 17 View dependent rendering con usuario por debajo de la cámara

En la siguiente figura (Figura 18) se muestra el efecto de usar el *head tracker* para realizar la traslación, sin corregir el *frustum* para obtener el efecto *parallax* como se explicó en el punto 3.2.3. El resultado es que simplemente se desplaza la cámara.

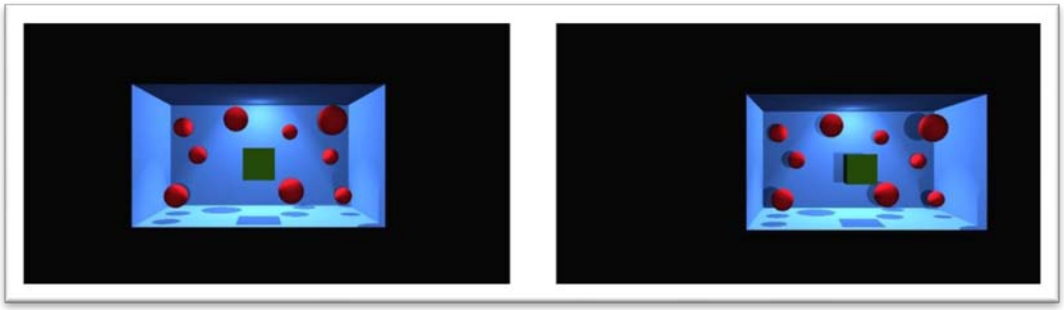

Figura 18 View Dependent Rendering sin corrección del *frustum*. En la imagen de la derecha se ha desplazado la cabeza hacia la izquierda

En la siguiente serie (Figura 19) se muestra la influencia del parámetro que nos permite manipular el valor Z de la cabeza sin necesidad de moverla. Rebajando dicho parámetro se consigue acentuar la profundidad. Dicha acentuación es el efecto que se produce también al acercarse a la pantalla. El efecto es lógico ya que imaginemos, por ejemplo, que nos acercamos a un caja física dispuesta de la misma forma que nuestra Cornell Box, a medida que nos acerquemos podremos ver más extensión de las paredes de la caja. Un valor muy alto del parámetro disminuye la sensación de inmersión y un valor demasiado bajo distorsiona en exceso la escena.

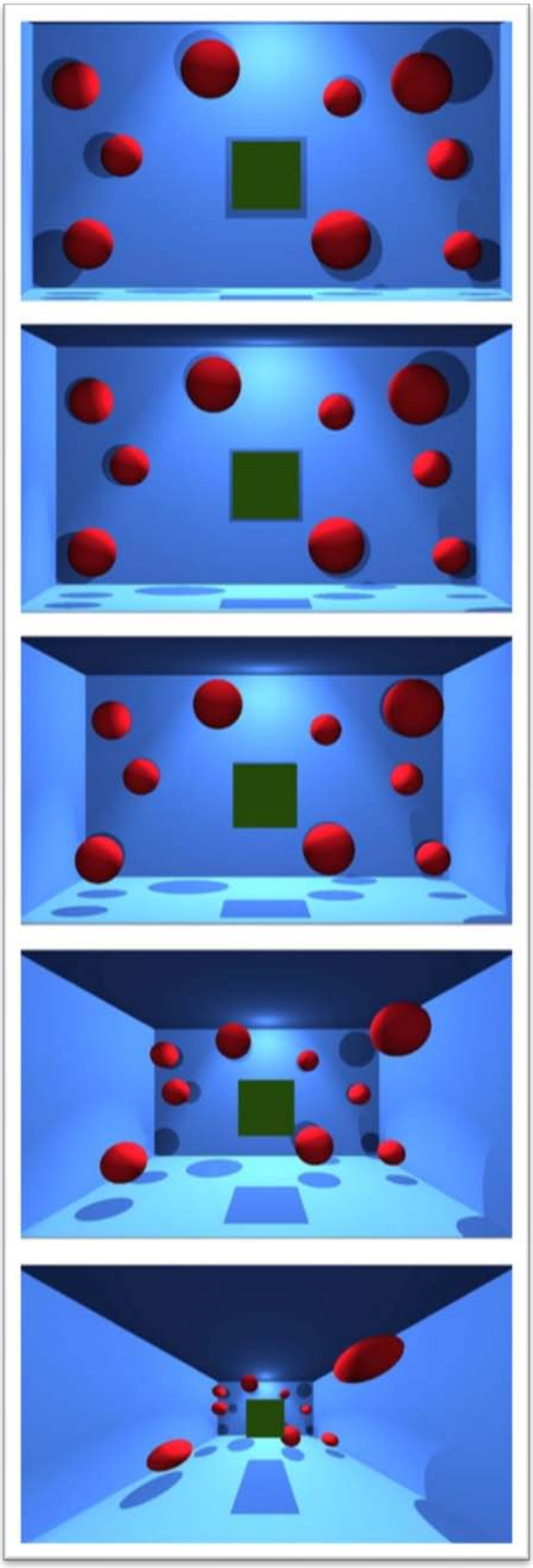

Figura 19 Aumentando la profundidad provocado por la manipulación del parámetro de control

La utilidad se comporta correctamente. Sin embargo presenta algunas limitaciones.

- La principal limitación de ésta técnica es que el marco es fijo. Esto requiere posicionar nuestra escena de tal manera que se ajuste al marco y después desplazar la cámara para ajustarla a éste también. El marco se encuentra en el plano Z=0 y su altura y ancho dependen de la relación de aspecto de la aplicación.

- La cámara del Wiimote tiene un ángulo de visión de 45 grados y detecta marcadores a un máximo de 5 metros de distancia. Sin embargo no se contempló que los marcadores leds IR también tenían su haz de apertura, es decir que su luz es direccional y no puede detectarse mirando de perfil el led. Para los leds IR utilizados el ángulo de visión es de 40 grados. Esto limita el giro de la cabeza del usuario. La solución es adquirir leds IR con un haz de apertura mayor, sin embargo una apertura mayor implica una distancia menor. En definitiva sería necesario estudiar las especificaciones de los distintos leds IR en el mercado y elegir los más adecuados a la instalación. O bien instalar varios leds IR a cada extremo de la cabeza a modo de abanica de igual forma que están instalados en la *sensor bar* de la Wii.

- La cámara del Wiimote y en general cualquier cámara digital ofrece menos precisión para hacer *tracking* a medida que nos alejamos de ésta. Este hecho no supone un punto negativo para la aplicación, dado que no se necesita obtener la posición exacta de la cabeza del usuario, ya que con una aproximación ya podemos obtener el efecto deseado.

Y como puntos positivos remarcables:

- Implementar un *view dependent rendering* normal (que realiza traslaciones y rotaciones) simplemente hubiera supuesto liberar las manos del usuario del teclado y el ratón para manipular la cámara. Sin embargo el *view dependent rendering* implementado ofrece un factor diferencial que lo haría complicado manipularlo mediante ratón y teclado, es por ello que se presenta como una mejor opción de *view dependent rendering*.

- El uso de marcadores y cámara infra-roja nos proporciona los resultados deseados en cuanto a fluidez. Se pueden realizar movimientos extremadamente rápidos sin que se pierda la posición de la cabeza gracias a los 100 *fps* capturados por la cámara del Wiimote.

A pesar de que la utilidad implementada nos da los resultados esperados, el nivel de interacción que proporciona por si sola es limitado. Es por ello que se decide estudiar otras vías para ofrecer más interacción, ampliando la primera utilidad o creando una segunda utilidad.

PARTE 2 - MOTION CONTROLLER

4. Pre-análisis

En este punto se explicará el proceso de análisis que derivo en el desarrollo de la segunda utilidad. En un primer momento se consideraron varias opciones basadas en añadir funcionalidades a la primera utilidad.

Dichas opciones se enumeran a continuación:

- Añadir un tercer marcador para obtener la orientación de la cabeza.
- Añadir un segundo Wiimote como cámara y emplear técnicas de estereopsis para obtener la correspondencia de los marcadores entre las dos cámaras. Esto se traduce en obtener más grados de libertad con menos marcadores:
    - 1 marcador: 3 grados de libertad (traslación en los tres ejes).
    - 2 marcadores: 6 grados de libertad (traslación y orientación completa).

La opción de la estereopsis resultó interesante en un primer momento. Si se planteaba como ampliación de la primera utilidad nos permitía obtener un *head tracker* con 6 grados de libertad, en lugar de 4. Sin embargo un sistema así no es más que una substitución limitada de un ratón y un teclado, ya que nos libera las manos de éstos pero no podemos realizar giros completos ni traslaciones muy extensas. Por este motivo se consideró como opción más interesante el uso de un marcador con 3 grados de libertad.

Dado un único marcador se planteó su utilidad en el campo de la interacción. Trasladar el marcador en el espacio real y mapear dicha traslación en el entorno virtual fue la opción elegida a priori. De este modo se podría usar el marcador para coger objetos virtuales y trasladarlos por la escena libremente.

La opción de la estereopsis parecía prometedora, sin embargo se observaron varios inconvenientes relativamente importantes. A diferencia de la primera utilidad en este caso si se requería de precisión y la combinación del reducido tamaño de los marcadores con la estereopsis no nos la podían proporcionar (Figura 20). El segundo inconveniente era relativo a usar la estereopsis para obtener la posición del marcador y a la vez seguir usando el *head tracking.* El *head tracking* implementado requiere posicionar la cámara en el centro horizontal de la pantalla y a la altura e inclinación deseada verticalmente (encima o debajo de la pantalla y enfocando al usuario). La estereopsis sin embargo requiere posicionar los Wiimotes de tal forma que obtengamos el mayor área, normalmente equidistantes del centro de la pantalla. Por lo tanto alejados del centro. El uso de marcadores presenta un gran inconveniente si queremos combinar utilidades, ya que no es trivial identificar que marcador pertenece a la cabeza y cual al que es empleado como puntero tridimensional. Y el último problema es el ángulo del haz de luz de los leds IR, que limita tanto las traslación como la rotación de éstos.

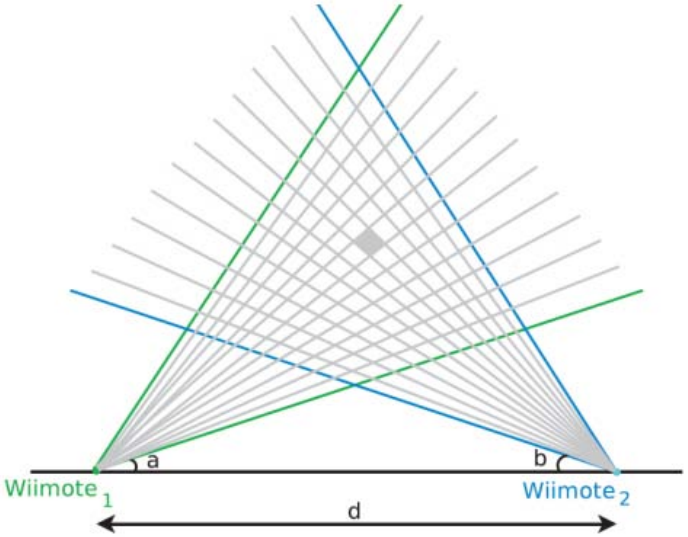

Figura 20 Wiimotes como cámaras para estereopsis. A mayor distancia tenemos menos precisión

Llegados a este punto se descartó la opción de la estereopsis y se empezó a estudiar el Wiimote no como cámara IR, sino como *motion controller.*

5. Extensión de las capacidades interactivas del Wiimote

Con la opción de la estereopsis descartada, se estudiaron las capacidades del Wiimote como *motion controller* (control de movimiento o control gestual). El objetivo del estudio era determinar si empleando un Wiimote podíamos conseguir resultados iguales o superiores a los que conseguiríamos usando estereopsis y un marcador, pero sin los inconvenientes de este sistema. Es decir, conseguir mapear la posición del Wiimote real en un objeto virtual y obtener de este modo un puntero tridimensional preciso y que no interfiriera con el *head tracker* implementado.

5.1 El Wiimote

En diciembre de 2006 Nintendo lanzó al mercado su nueva consola de videojuegos, la Nintendo Wii. A nivel de gráficos la consola no daba un gran salto con respecto a su antecesora, la Nintendo GameCube. La característica más relevante era su mando inalámbrico, el Wiimote (Figura 21). Éste dispone de una cámara infra-roja, que en combinación con la *sensor bar* incluida en la consola, nos permitía usar el mando como un dispositivo apuntador. Pero la característica realmente relevante era que el mando era capaz de detectar la aceleración e inferir el movimiento del mando. Esto último se tradujo a nivel de jugabilidad en substituir las clásicas pulsaciones de botones por movimientos del mando.

Figura 21 Vista frontal del Wiimote

En dispositivos portátiles ya existían distintos *tilt sensors* que ofrecían la posibilidad de interactuar inclinando el dispositivo portátil. Pero Nintendo decidió apostar por esta tecnología para su consola de sobremesa. La consola incluía un juego para demostrar el potencial de esta novedosa tecnología, Wii Sports, donde se podía jugar a tenis, beisbol, golf,

bolos y boxeo. De este modo podíamos, por ejemplo, jugar a tenis moviendo el mando como si fuera una raqueta. Sin embargo la raqueta virtual y el resto de artilugios virtuales no se movían en el entorno virtual de la misma forma que nosotros movíamos el mando. El motivo de esta decepción es debido a las limitaciones del hardware base del Wiimote, es decir los acelerómetros.

A continuación profundizaremos más sobre el hardware del Wiimote y como este condiciona las limitaciones en cuanto a control de movimiento.

5.1.1 Limitaciones del hardware

Además de los típicos botones de los que dispone cualquier controlador para videojuegos, de la vibración y del altavoz (con un sonido de pésima calidad), las características distintivas del Wiimote son su cámara infra-roja ubicada en la parte superior y su acelerómetro de 3 ejes ADXL330 ubicado en su interior a la altura del botón A. La combinación de la cámara, el acelerómetro y el bluetooth hacen del Wiimote un dispositivo de gran interés para los desarrolladores de software en el ámbito de las tecnologías interactivas.

La cámara infra-roja del mando se usa para detectar los leds IR posicionados en una barra que incorpora la consola para colocarla encima o debajo del televisor. Dicha barra recibe el nombre de *sensor bar* a pesar de que realmente no incorpora ningún tipo de sensor. Mediante la detección de los leds de la barra se usa el Wiimote como un dispositivo apuntador.

El acelerómetro ADXL330 funciona igual que cualquier otro acelerómetro, como por ejemplo el que incorpora el *iPhone* o el *iPod Touch* . Éste detecta la aceleración en los 3 ejes (ya que en realidad son 3 acelerómetros, uno por eje) midiendo la fuerza G que actúa sobre estos. La fuerza G no es una medida de fuerza, sino una medida intuitiva de aceleración. Una aceleración de 1G es equivalente a la gravedad en la tierra (9.8 m/s^2) y el ADXL330 puede medir en un rango de ±3G. En función de las fuerzas G que actúan sobre cada acelerómetro se puede medir la dirección y velocidad del desplazamiento del Wiimote.

Los acelerómetros también pueden usarse para deducir la orientación en relación al mundo cuando el Wiimote está estático. Sin embargo únicamente con los acelerómetros no es posible conocer totalmente la orientación, tan solo se puede determinar el *pitch* (cabeceo en castellano) y el *roll* (alabeo) relativos al mundo. Para determinar el *yaw* (guiñada) se usa la cámara infra-roja apuntando a la *sensor bar*. En definitiva los acelerómetros no son una opción adecuada para medir la orientación.

Ya que no es aconsejable medir la orientación mediante los acelerómetros, centrémonos en el uso de éstos para calcular la posición. El cálculo de la posición del Wiimote mediante los acelerómetros tampoco es una medida aconsejable por básicamente dos factores. El primero es la falta de precisión de los acelerómetros y el segundo es el error que acumula el cálculo de la posición. Dicho cálculo se fundamenta en integrar los datos de los acelerómetros a lo largo del tiempo. Integrarlos una primera vez nos permite obtener la

velocidad y integrarlos de nuevo la posición, sin embargo el hecho de integrar de forma discreta, sumado a la imprecisión inherente de los acelerómetros hace que se acumule un error en la estimación de la posición a lo largo del tiempo. Esto quiere decir que si partimos de una posición inicial y trasladamos el Wiimote tendremos una estimación de la posición bastante fiable en los primeros segundos, pero a medida que avance el tiempo la estimación será cada vez más errónea. También perderemos fiabilidad si se realizan movimientos muy rápidos y con repentinos cambios de orientación o dirección. En definitiva, los acelerómetros tampoco son aconsejables para medir la posición en el espacio del Wiimote.

Las carencias del que supone el uso del acelerómetro se refleja claramente en muchos juegos del catálogo de la Nintendo Wii. Donde una acción que podría ser perfectamente mapeada en un botón se realiza mediante un movimiento del mando en alguna dirección. Este mapeo recibe el nombre de *waggle* que quiere decir oscilación o meneo.

Llegados a este punto podríamos haber concluido que el Wiimote no cumplía los requisitos que precisábamos para nuestra utilidad. Sin embargo seguimos profundizando en el tema y cambiamos de opinión.

En el verano de 2009 Nintendo puso a la venta el Wii Motion Plus. Éste consiste en un dispositivo que se conecta en la base del Wiimote ampliando las capacidades de detección de movimiento de éste. Nintendo promocionó su nuevo producto con el eslogan "*true motion 1:1".* El 1:1 hace referencia a que un movimiento realizado con el Wiimote usando el Wii Motion Plus se refleja del mismo modo en el juego. Es decir, adiós al *waggle*.

Sin embargo el Motion Plus, a pesar de mejorar de forma notoria las capacidades del Wiimote como controlador de movimiento, no hace posible el movimiento 1:1 prometido. A continuación analizaremos que mejoras aporta al Wiimote y en qué consiste su tecnología.

5.1.2 Wii Motion Plus

El Motion Plus contiene en su interior 2 giroscopios. Uno es un IDG-600 de 2 ejes y el otro un X3500W de 1 eje, que nos permiten obtener el giro del Wiimote en los 3 ejes. Sin embargo los giroscopios no nos indican cuanto gira el objeto sino la velocidad angular en cada eje. Para obtener hacía donde gira el Wiimote se debe integrar de forma discreta la velocidad angular y obtener así la traslación angular en un espacio de tiempo. Por lo tanto el Motion Plus sirve únicamente para obtener la orientación del Wiimote y no nos proporciona ningún tipo de información sobre su posición.

Cuando jugamos al juego lanzado junto al Motion Plus, el Wii Sport Resort, tenemos una primera impresión de que el avatar (en la Wii recibe el nombre de Mii) hace exactamente lo que nosotros hacemos. Ejemplo de ello lo encontramos en el juego de combates con espada del Wii Sport Resort (Figura 22) donde los espadazos de nuestro Mii se realizan con el mismo ángulo que los nuestros. Además también podemos realizar estocadas y cuando levantamos el Wiimote por encima de nuestra cabeza, para realizar un corte vertical largo, el Mii también lo

hace. Este último ejemplo de levantar el Wiimote sobre nuestra cabeza nos servirá para revelar el truco del Motion Plus de forma breve.

Figura 22 Combate con espadas en Wii Sports Resort

Cuando trasladamos el Wiimote sobre nuestra cabeza para atacar con éste como si fuera una espada variamos la orientación de éste de tal forma que apunta hacia atrás. Así pues, ¿que podría pasar si apuntásemos con el Wiimote hacia atrás sin levantarlo sobre nuestra cabeza? Pues el resultado es exactamente el mismo que si lo levantásemos, vemos el Mii levantando la espada por encima de su cabeza. El truco consiste sencillamente en asociar la componente rotacional del movimiento humano con una traslación, es decir, presuponer que en un escenario concreto una rotación detectada siempre irá acompañada de cierta traslación, como es el caso de la espada sobre nuestra cabeza.

La solución que nos da el Motion Plus, a pesar no ser el ansiado movimiento 1:1 es claramente una solución superior a la que nos proporciona el Wiimote por sí solo. Sin embargo los giroscopios, al igual que los acelerómetros son ciegos. Esto quiere decir que los acelerómetros no saben la posición actual sino que, como explicamos en el punto anterior, partiendo de una posición inicial podemos estimar en los instantes de tiempo sucesivos la nueva posición en base a la velocidad obtenida de integrar la aceleración. Los giroscopios funcionan del mismo modo, se debe partir de una orientación conocida e integrando la velocidad a lo largo del tiempo sabremos cuanto ha rotado el Wiimote.

Los giroscopios nos permiten realizar rotaciones realmente suaves, pero al igual que los acelerómetros acumulan un error a lo largo del tiempo que hace que su estimación de la orientación sea cada vez menos precisa desde la última orientación conocida. Volviendo al juego de combates con espada, antes de empezar la partida se nos obliga a apuntar a la pantalla y apretar un botón para empezar a jugar. Es en este momento cuando se obtiene la orientación del Wiimote, no mediante los giroscopios, sino mediante los acelerómetros y la cámara como se explicó en el punto anterior. Es decir antes de empezar a jugar obtenemos una medición válida de la orientación que nos sirve como estado inicial. Esto haría pensar que entonces las partidas tienen que ser muy breves para que no empecemos a notar la desorientación del Motion Plus a lo largo del tiempo, sin embargo está desorientación puede

ser subsanada. Para ello se aprovechan los momentos de la partida en los que no estamos moviendo el Wiimote para reajustar la orientación usando los acelerómetros y la cámara de igual modo que se usan antes de empezar a jugar. Usando una vez más como ejemplo el juego de combate de espadas, si empezamos a mover la espada con movimientos rápidos y aleatorios y nos detenemos, veremos como la espada no apunta a donde apuntamos nosotros con el Wiimote. Pero justo en ese mismo instante también veremos como la espada empieza a rotar suavemente hasta llegar a la orientación correcta, proporcionada por los acelerómetros (*pitch* y *roll)* y la cámara del Wiimote si estamos apuntando a la pantalla (*yaw).* En resumen, usar el Wiimote con el Motion Plus nos permitiría obtener una buena aproximación de la orientación siempre y cuando la fuéramos recalibrando mediante los acelerómetros y la cámara IR.

Llegados a este punto vimos viable el uso del Wiimote para obtener la orientación y mapearla en un objeto del mundo virtual, pero estimar de forma fiable la posición seguía siendo un problema intratable con el Wiimote. Por este motivo se decidió estudiar un dispositivo que, a pesar de no estar todavía disponible en el mercado, si cubría todos los requisitos que se esperan de un *motion controller.*

5.2 El PlayStation Move

El PlayStation Move (Figura 23) es el control gestual de Sony para su consola de videojuegos PlayStation3. Éste fue presentado en la Electronic Entertainment Expo del 2009 y será puesto a la venta en Europa el 15 de setiembre. Su diseño es muy similar al del Wiimote sin embargo su tecnología es superior. En su interior podemos encontrar un acelerómetro de 3 ejes al igual que en Wiimote. También podemos encontrar un giroscopio de 3 ejes al igual que el Wii Motion Plus. Estos dos componentes ya lo igualan en prestaciones al Wiimote con Motion Plus, pero hay un tercer componente en el interior de PlayStation Move que marca una primera diferencia.

Figura 23 PlayStation Move

Este elemento recibe el nombre de magnetómetro y permite detectar la magnitud y dirección del campo magnético de la tierra. Sabiendo de donde proviene el campo magnético o lo que es lo mismo, el PlayStation Move puede conocer su orientación de forma directa. Es decir, los magnetómetros no son sensores ciegos como los giroscopios y los acelerómetros. Sin embargo tiene un pequeño inconveniente. Son muy sensibles a materiales ferromagnéticos cercanos, ya que éstos alternan el campo magnético y por lo tanto su percepción. Por este motivo la orientación en el PlayStation Move no nos la dan los acelerómetros, los giroscopios ni los magnetómetros, sino el uso combinado de los 3 sensores.

En el Wiimote teníamos que los datos de la orientación proporcionados por giroscopios era recalibrados mediante los acelerómetros, que reajustaban el *pitch* y el *roll,* y la cámara en combinación de la *sensor bar,* que reajustaba el *yaw.* En el PlayStation Move se recalibran los datos de la orientación proporcionados por los giroscopios con la combinación de los acelerómetros y los magnetómetros. Con los acelerómetros se compensa la posible distorsión magnética de los magnetómetros y como los magnetómetros nos dan la orientación de forma directa usamos ésta para recalibrar la orientación proporcionada por los giroscopios. En resumen el PlayStation Move nos da una medición fiable de la orientación sin necesidad de recalibrarse con un elemento externo como hace el Wiimote con la cámara y la *sensor bar*.

Pero los magnetómetros no son el único factor diferencial entre el PlayStation Move y Wiimote. El factor diferencial más importante entre estos dos controles gestuales es que a diferencia del Wiimote, el PlayStation Move si nos permite conocer de forma precisa la posición de éste en el espacio. Esto se consigue mediante dos componentes uno integrado en el PlayStation Move y otro externo a éste. El elemento externo es la cámara PlayStation Eye que permite capturar 60 imágenes por segundo a 640x480 píxeles de resolución. Y el elemento integrado en el PlayStation Move es una esfera iluminada, a modo de marcador, ubicada en el extremo del dispositivo. La esfera se ilumina de un determinado color y la cámara procesa cada imagen buscando ese color y obteniendo la posición de la esfera en el plano. Dado que el tamaño de la esfera es conocido por el software, se puede calcular la distancia en base a el tamaño de la "mancha" de color encontrada por la cámara.

Se pueden dar situaciones en que la esfera sea ocluida, en estas situaciones se usan los acelerómetros para estimar la posición ya que las oclusiones serán durante un periodo breve de tiempo los acelerómetros tendrán un error bajo en la estimación. Los motivos por los que la esfera está iluminada son varios. El principal es porque esto permite detectarla independientemente de las condiciones de iluminación del entorno donde se esté jugando. El cambio de color permite adaptar el filtrado. Es decir, que si la esfera se ilumina azul y en el entorno existe mucho azul que pueda interferir en la fiabilidad de la detección, la esfera se iluminará de un color distinto y se detectará este color en lugar del azul.

5.3 Conclusiones

Una vez estudiado el funcionamiento del Wiimote y el PlayStation Move se consideró una opción viable implementar un híbrido de ambos controladores, o lo que es lo mismo,

extender las capacidades del Wiimote asimilándolas a las del PlayStation Move. El proceso fue dividido en dos bloques. En primer lugar obtener la orientación del Wiimote y en segundo lugar obtener la posición. Para la primera tarea se consideró imperativo el uso del Motion Plus. En cuanto a la segunda tarea se barajaron y testearon varias opciones que serán detalladas en el capítulo 7.

6. Wii Motion Plus: obtener la orientación

6.1 Análisis

En este punto se estudiará como obtener los datos del Wii Motion Plus y procesarlos para conseguir la orientación del Wiimote. Este proceso se dividirá en varios apartados. Primero se estudiarán que datos nos proporciona la librería *WiiYourself!*. Una vez estudiados los datos se presentará el proceso de calibración de los giroscopios. A partir de este punto ya seremos capaces de obtener la orientación del Wiimote partiendo de un estado inicial, sin embargo la orientación estimada a lo largo del tiempo irá perdiendo precisión, sobretodo ejecutando giros rápidos. Por este motivo se analizarán las distintas opciones para reajustar la orientación, todas ellas fundamentadas en el uso de los acelerómetros del Wiimote.

6.1.1 WiiYourself!: obtener la velocidad angular

Como se mencionó brevemente en el punto 3.3.1 la librería que se eligió finalmente para implementar las dos utilidades fue WiiYourself!. Ésta fue elegida principalmente por dos motivos. Tiene soporte para el Motion Plus y está programada en C++.

La librería incluye un método para detectar si hay conectado un Motion Plus al Wiimote y activarlo si es así. Una vez activado podemos acceder a la información de los 3 giroscopios de dos formas. La primera es acceder a los datos RAW y la segunda es acceder a los datos procesados, que vendrían a ser directamente las velocidades angulares de cada giroscopio expresadas en radianes/segundo.

En un primer momento se usaron las velocidades angulares, pero estas no estaban correctamente calibradas provocando un lento y continuado cambio de la orientación del objeto virtual, a pesar de tener el Wiimote completamente parado. Por ello se descartó el uso de los datos procesados y se decidió analizar en qué consistían los datos RAW y como convertir éstos en una mejor aproximación de las velocidades angulares que la ofrecidas por la librería.

Los datos *RAW* de cada giroscopio eran ligeramente diferentes. Con el Wiimote estático los valores no eran ni mucho menos cercanos al cero. Todos ellos estaban entre 7900 y 8400 y las decenas y unidades variaban constantemente. No entraremos a explicar en detalle el porqué de dichos valores, simplemente indicar que tienen relación con el conversor analógico-digital y el voltaje subministrado al Motion Plus.

El proceso de conversión de los datos RAW a velocidad angular de la librería es muy escueto. Para obtener la velocidad angular se obtiene el dato RAW y se le resta 8063 que actúa como valor cero o *bias* para los 3 giroscopios. El valor obtenido es multiplicado por un factor de escala para obtener la velocidad angular. En este proceso encontramos dos problemas. El primero es tomar 8063 como valor cero para todos los giroscopios. Lo correcto sería determinar un valor cero más preciso para cada giroscopio. Es decir, el valor que tienen

normalmente cada giroscopio en estado de reposo. Esto es lo que lograremos con el proceso de calibración. El segundo problema lo genera el factor de escala. El factor de escala funciona a modo de conversor, transformando las unidades a velocidad angular. Sin embargo el factor de escala no es un valor fijo, éste depende del modo en que se encuentre el giroscopio en ese momento. Los modos son dos: lento y rápido. En el modo lento el factor de escala es de 0.05 y en el modo rápido 0.25. Esto permite que por ejemplo una lectura de 20 unidades con respecto al cero sea equivalente a un grado por segundo en el modo lento y 5 grados por segundo en el modo rápido. Esto permite no perder precisión cuando se realizan movimientos rápidos. Sin embargo testeando la aplicación se observó que el factor de escala en ambos modos era muy pequeño, teniéndolo que re-escalar para obtener el giro del objeto virtual igual al del Wiimote.

En resumen para obtener la velocidad angular de forma más precisa se cambió el factor de escala y se ideo el proceso de calibración que a continuación explicaremos para obtener unos ceros más correctos que los predefinidos por la librería de 8063. La modificación del factor de escala supuso necesitar obtener los dos modos de los giroscopios (lento y rápido), por este motivo se modificó y recompiló la librería WiiYourself! 1.15 haciendo público ese dato.

6.1.2 Calibración de los ceros

El proceso de calibración tiene como objetivo encontrar unos ceros más correctos. El problema de WiiYourself! tomando 8063 como valor cero para los tres giroscopios es que si tenemos el Wiimote estático y las lecturas en un giroscopio son por ejemplo de 7900, al restar el valor cero de 8063 y escalar obtendremos que en ese eje existe una velocidad angular y por lo tanto el objeto virtual girará cuando el real no.

Para encontrar una buena estimación de los ceros de los giroscopios se empleará un método muy sencillo, pero muy efectivo. Este consiste en tomar un número determinado de lecturas para cada giroscopio con el Wiimote completamente quieto, por ejemplo 1000 y una vez obtenidas hacer la media. Esta media será un valor mucho más correcto que el 8063. A mayor número de lecturas mejor será el cero calculado, pero el proceso de calibración tomará más tiempo.

Con el Motion Plus calibrado seremos capaces de obtener una estimación de la orientación bastante precisa y mantener el Wiimote quieto y que el objeto virtual también lo esté. Sin embargo, cuanto mayor sea el tiempo transcurrido desde nuestro estado inicial, más grande será el error de la estimación, es por ello que deberemos reajustar la orientación en base a la orientación que en ciertos momentos podremos calcular con los acelerómetros.

6.1.3 Acelerómetros para calcular el *pitch* y el *roll*

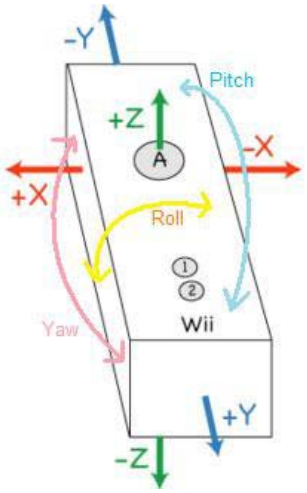

Figura 24 Acelerómetros X Y Z representados en el Wiimote

Un acelerómetro mide la aceleración que se produce sobre un eje determinado. Cuando el Wiimote no está siendo trasladado la única aceleración que percibe es la de la gravedad. En el Wiimote podemos encontrar 3 acelerómetros (Figura 24), uno por cada eje. Si el Wiimote se encuentra plano con los botones mirando hacia arriba una lectura de los valores (X,Y,Z) retornados por los acelerómetros sería (0,0,1), siendo estos valores expresados como fuerza G, es decir 1 equivale a la aceleración de la gravedad. Si empezásemos a girar el Wiimote hacia la derecha veríamos como el valor Z iría decrementando . Cuando el mando estuviera completamente ladeado (aplicado un cuarto de vuelta), el valor de los acelerómetros pasaría a ser (1,0,0). Si seguimos girando el Wiimote el valor de X decrementará llegando a valer 0 cuando el Wiimote esté tumbado con los botones hacia abajo. Pero el valor de Z habrá incrementado en magnitud pero con signo negativo hasta valer -1. En esta situación la lectura de los acelerómetros sería (0,0,-1). Otro cuarto de vuelta nos daría los valores (-1,0,0) y con un cuarto de vuelta más volveríamos al estado inicial.

De este pequeño experimento con los acelerómetros se pueden determinar varías cosas. Girando el Wiimote sobre el eje Y, es decir haciendo un *roll,* la lectura del acelerómetro del eje Y ha permanecido nula. Eso es debido a que el eje Y permanece perpendicular a la gravedad cuando se realiza un *roll*. La relación entre la aceleración percibida por los acelerómetros X y Z nos permite determinar el *roll* cuando el Wiimote solo percibe gravedad, es decir, cuando las lecturas de los 3 acelerómetros es igual o inferior a 1. De igual modo podríamos obtener el *pitch* relacionando las lecturas entre los acelerómetros Y y Z. Sin embargo es imposible obtener el *yaw*, ya que al rotar sobre el eje Z, los acelerómetros de X e Y son perpendiculares a la gravedad, dando ambos valor 0.

$$\operatorname{atan2}(y,x) = \begin{cases} \arctan(\frac{y}{x}) & x > 0 \\ \pi + \arctan(\frac{y}{x}) & y \geq 0, x < 0 \\ -\pi + \arctan(\frac{y}{x}) & y < 0, x < 0 \\ \frac{\pi}{2} & y > 0, x = 0 \\ -\frac{\pi}{2} & y < 0, x = 0 \\ \text{undefined} & y = 0, x = 0 \end{cases}$$

Figura 25 Función atan2

Para obtener el ángulo en radianes del *pitch* y el *roll* se usa la función trigonométrica atan2 (Figura 25), atan(X,Z) para el *roll* y atan(Y,Z) para el *pitch.*

Ahora que sabemos determinar el *pitch* y el *roll* gracias a los acelerómetros, analizaremos las distintas soluciones estudiadas para corregir la orientación estimada por los giroscopios mediante los acelerómetros.

6.1.4 Corrección de la orientación mediante los acelerómetros

El primer método implementado para corregir la orientación consistió en reajustar la orientación en los momentos que el Wiimote solo percibiera aceleración de la gravedad, es decir, que la lectura de los 3 acelerómetros fuera igual o inferior a 1. Como segunda condición se requirió que la aceleración en Y o en X fuera 0. Si la aceleración era 0 en Y quería decir que podíamos obtener el *roll* y si lo era en X podíamos obtener el *pitch* mediante el atan2 de la lectura de los acelerómetros en los otros dos ejes.

Así pues, cuando se daban ambas condiciones se reajustaba el *roll* o el *pitch* del objeto virtual. La teoría tiene sentido, sin embargo a la práctica este método presenta varios problemas. El primero y más evidente es el hecho de que es extremadamente tosco. Es decir, la rotación se irá calculando mediante los giroscopios hasta que se den las condiciones estipuladas. En ese momento puede ser que el *roll* o el *pitch* tengan una desviación muy notable que al ser corregida de la impresión de un cambió de rotación instantáneo, lo cual puede resultar confuso y molesto. El segundo inconveniente es que los acelerómetros no dan realmente cero cuando son perpendiculares a la gravedad, sino que dan valores cercanos a cero. Por ello se delimitó un rango para aplicar la condición del cero. Es decir que cuando por ejemplo la Y estaba cerca del 0 reajustábamos el *roll.* El hecho de usar un rango provocaba que, mientras el valor estuviera dentro de ese rango, el ángulo se reajustaba constantemente. Es decir que era como prescindir de los giroscopios dentro de ese rango. El problema de esto es que el ángulo determinado por los acelerómetros es algo ruidoso, lo que visualmente se aprecia como un temblor constante del *roll* o el *pitch.*

Dado que los resultados no fueron suficientemente admisibles se decidió buscar alternativas a la forma de fusionar los datos. La mayoría de la literatura coincidía en que el filtro Kalman [7] era la mejor opción para fusionar las lecturas de ambos sensores.

El filtro Kalman es un filtro iterativo que requiere dos cosas. En primer lugar unos datos de entrada. Estos datos son convertidos en una predicción usando únicamente cálculos lineales. Para esto último se requiere un modelo lineal del problema a tratar. En segundo lugar se necesitan otros datos como entada. A cada iteración, el filtro Kalman cambiará las variables de nuestro modelo lineal de tal forma que se parezcan poco a poco a los de la segunda entrada.

Aplicado a nuestro caso los primeros datos de entrada serían las velocidades angulares de los giroscopios. El modelo lineal sería el que convertiría las velocidades angulares en ángulos. Los segundos datos de entrada serían los ángulos determinados por los acelerómetros. Es decir, los ángulos calculados mediante los giroscopios irían asimilándose de forma suave a los calculados por los acelerómetros de igual manera que explicamos en el punto 5.1.2 sucedía en la Wii.

El filtro Kalman se compone de dos fases. La predicción y la actualización (Figura 26). En la fase de predicción se usa el estado estimado en la anterior iteración para estimar el estado en la iteración actual. Este estado predicho es conocido como el estado *a priori*, porque es una estimación del estado en la iteración actual, ya que no incluye observaciones del estado actual real (los segundos datos de entrada). En la segunda fase, la de actualización, la predicción *a priori* se combina con la observación para refinar la estimación del estado. El estado refinado recibe el nombre de estado *a posteriori*. En cada iteración se ejecutará la fase de predicción y de actualización, sin embargo si en la iteración no se dispone de observaciones, únicamente se ejecutará la predicción.

Predict

Predicted (*a priori*) state estimate	$\hat{\mathbf{x}}_{k\mid k-1} = \mathbf{F}_k\hat{\mathbf{x}}_{k-1\mid k-1} + \mathbf{B}_k\mathbf{u}_k$
Predicted (*a priori*) estimate covariance	$\mathbf{P}_{k\mid k-1} = \mathbf{F}_k\mathbf{P}_{k-1\mid k-1}\mathbf{F}_k^{\mathrm{T}} + \mathbf{Q}_k$

Update

Innovation or measurement residual	$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}_k\hat{\mathbf{x}}_{k\mid k-1}$
Innovation (or residual) covariance	$\mathbf{S}_k = \mathbf{H}_k\mathbf{P}_{k\mid k-1}\mathbf{H}_k^{\mathrm{T}} + \mathbf{R}_k$
Optimal Kalman gain	$\mathbf{K}_k = \mathbf{P}_{k\mid k-1}\mathbf{H}_k^{\mathrm{T}}\mathbf{S}_k^{-1}$
Updated (*a posteriori*) state estimate	$\hat{\mathbf{x}}_{k\mid k} = \hat{\mathbf{x}}_{k\mid k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k$
Updated (*a posteriori*) estimate covariance	$\mathbf{P}_{k\mid k} = (I - \mathbf{K}_k\mathbf{H}_k)\mathbf{P}_{k\mid k-1}$

Figura 26 Predicción y actualización del filtro Kalman

Para probar el funcionamiento del filtro Kalman se usó una implementación empleada para funcionar en placas Arduino y fue adaptada. Sin embargo dicha implementación presentó varios problemas. En primer lugar, como se puede apreciar en la figura 26, el filtro Kalman requiere configurar varios parámetros, todos ellos matrices. Dicha configuración complica excesivamente la configuración del comportamiento del filtro. En segundo lugar la implementación adaptada aplicaba un filtro Kalman para el *pitch* y otro para el *roll.* Tratar los ángulos de forma separada implicaba encadenar rotaciones. Esto puede producir un fenómeno conocido como *gimbal lock* (bloqueo del cardán) . Dicho fenómeno consiste en la pérdida de un grado de libertad al realizar ciertas rotaciones (Figura 27).

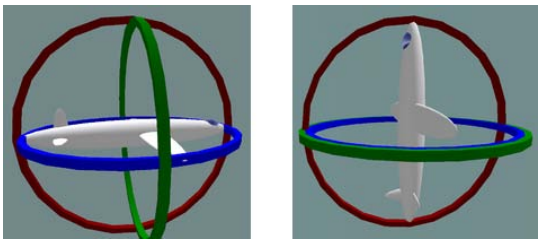

Figura 27 *Gimbal lock*

En la figura 27 se puede apreciar como al hacer un *pitch* de 90 grados los ejes del *roll* y el *yaw* se vuelven paralelos. En esta situación *roll y yaw* se convierten en la misma rotación quedándonos tan solo con dos grados de libertad. Realmente el termino *lock* (bloqueo) no es muy apropiado, ya que realmente no se bloquea ningún eje.

Para no sufrir *gimbal lock* se recomienda el uso de cuaterniones. Un cuaternión se considera una extensión de los números complejos a las 4 dimensiones. Éste es una tupla formada de 4 elementos (x,y,z,w). Cuando la suma de los cuadrados de los 4 elementos es igual a 1 el cuaternión puede ser usado para representar una rotación. A diferencia de los ángulos de Euler, que requieren concatenar las 3 rotaciones, con los cuaterniones se define una única rotación equivalente. Las rotaciones con cuaterniones no sufren *gimbal lock* porque son definidas con los 4 términos de éste. Es decir, que en el caso de producirse un bloqueo, los 4 términos permiten no bajar nunca de los 3 grados de libertad.

En resumen, la mejor opción era usar un filtro Kalman, pero usando cuaterniones. Para ello el filtro Kalman básico suponía una limitación dada su linealidad. Así que se analizaron versiones no lineares del filtro. En concreto se analizaron el *extended Kalman filter* [8] y el *unscented Kalman filter* [9]. Ambos resultaron difíciles de comprender, pero a pesar de ello se detectó dos grandes inconvenientes que hicieron que ninguno de los dos filtros fuera implementado. Los parámetros de ambos filtros eran matrices de dimensiones considerables, algunas de hasta cien términos, haciendo aún más compleja la configuración de los parámetros para ajustar el comportamiento del filtro. La dimensión de los parámetros suponía un segundo problema, ya que tales dimensiones requerían un mayor tiempo de cálculo, lo que hacía totalmente desaconsejable su uso para filtrado en tiempo real a altas velocidades de muestreo.

Llegados a este punto descartamos el filtro Kalman como método corregir la orientación y se pensó en dejar como final la primera solución a pesar de ser tosca. Sin embargo en un último intento de búsqueda se dio con la solución definitiva.

Sebastian O. H. Madgwick presenta en su publicación *An efficient orientation filter for inertial and inertial/magnetic sensor arrays* [10] un algoritmo que hace exactamente lo que queremos. En la descripción de éste se explica que ha diseñado un filtro aplicable a *IMUs* (*Inertial Measurement Units*) y a *MARGs (Magnetic, Angular Rate and Gravity).* Concretamente es aplicable a *IMUs* con acelerómetros y giroscopios de 3 ejes, al igual que nuestro Wiimote con Motion Plus. El filtro también es aplicable a MARGs, que son IMUs con un tercer componente, un magnetómetro, como por ejemplo el PlayStation Move.

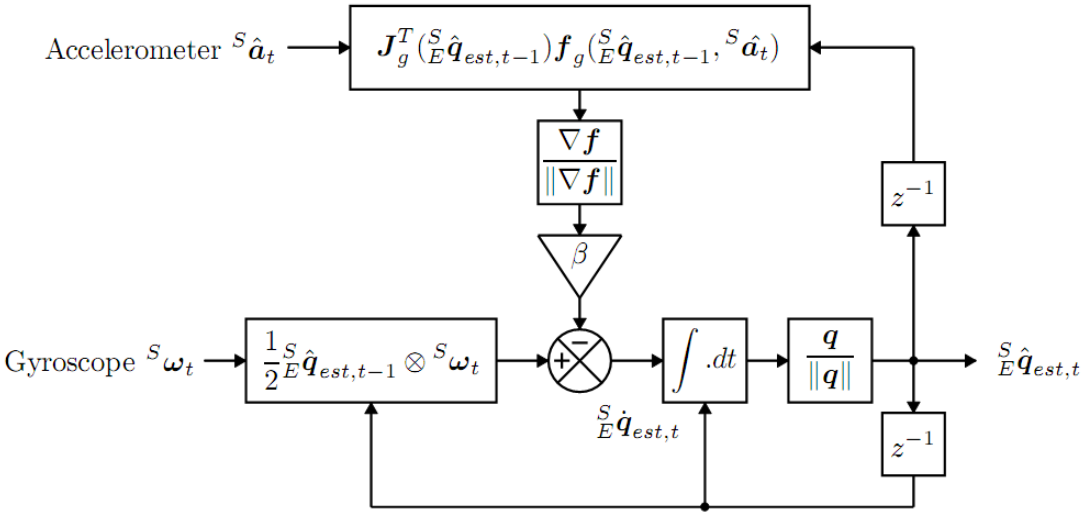

Figura 28 Diagrama de bloques del filtro de la orientación para *IMUs*

La solución propuesta por Madgwick (Figura 28), que a partir de este punto pasaremos a llamar *IMUfilter*, tiene 3 ventajas principales que claramente la hacen la mejor opción:

- Uso de cuaterniones.
- Tiempo de procesado mínimo. Tan solo emplea 109 operaciones aritméticas en cada iteración.
- Un único parámetro para configurar el filtro.

A diferencia del filtro Kalman, el *IMUfilter* no es un predictor, únicamente fusiona las dos entradas empleando técnicas como gradiente descendiente o la matriz Jacobiana (una matriz formada por derivadas parciales de primer orden de una función).

Para inicializar el filtro únicamente deberemos indicar la orientación inicial y el parámetro de configuración del filtro, al que llamaremos *beta*. Un valor nulo de *beta* anulará la entrada de los acelerómetros y un valor elevado dará más peso a éstos. Reajustándose la orientación de forma más rápida, pero provocando mucho más ruido una vez reajustada.

Dado que el filtro combina acelerómetros y giroscopios, únicamente se podrá reajustar el *pitch* y el *roll* del objeto virtual. Para reajustar el *yaw* se analizarán varias opciones más adelante.

6.2 Diseño

Dado que en el análisis ya hemos entrado a detallar todas las decisiones tomadas, en este punto únicamente esbozaremos los módulos que conformarán esta parte de la utilidad (Figura 29).

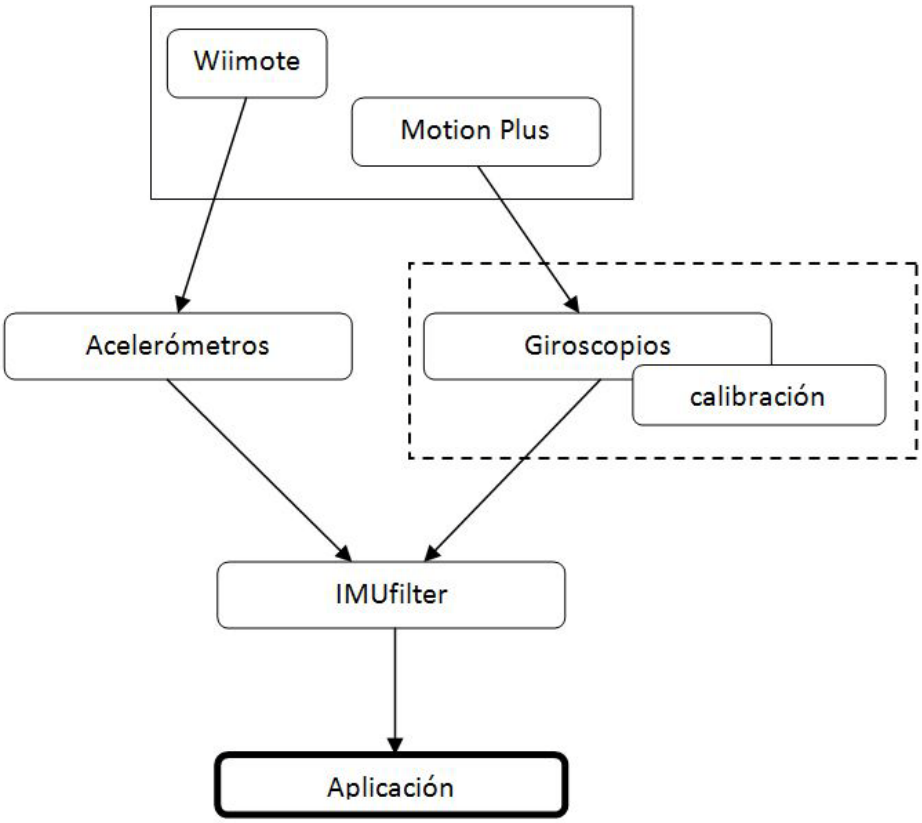

Figura 29 Bloques del diseño para obtener la orientación del Wiimote con Motion Plus

El proceso consistirá en obtener las lecturas de los acelerómetros y giroscopios y que estos sean procesados por el IMUfilter, que nos proporcionará la orientación corregida y en forma de cuaternion. Los datos proporcionados por los giroscopios deberán ser recalibrados en caso de que los ceros se alejen de los ceros estimados por el calibrador en el momento de la calibración. Los ceros serán iniciados a valores de ceros calculados en anteriores ejecuciones del calibrador, es decir, a priori no será necesario usar el calibrador. Sin embargo hay condiciones que harán distanciarse a los ceros de los ceros establecidos como aceptables en anteriores calibraciones. Estas condiciones son básicamente la variación de voltaje y de temperatura. En condiciones de temperatura distintas a las de cuando se realizó la calibración

los ceros serán ligeramente distintos. Esto podría darse por ejemplo sosteniendo el Wiimote durante un tiempo prolongado, transmitiendo de este modo nuestro calor corporal.

6.3 Implementación: módulos y funcionalidades

En este apartado se detallarán como han sido implementados los distintos módulos y sus funcionalidades, que nos permiten en conjunto obtener la orientación del Wiimote y mapearla en un objeto virtual.

Para la implementación de los módulos de la primera parte del *motion controller* se ha recuperado la clase **WiimoteManager** y se implementado en esta la detección y activación del Motion Plus y métodos para acceder a los valores de los giroscopios y los acelerómetros. También se han añadido a **WiimoteManager** las variables y métodos necesarios para el proceso de calibración.

Para el filtro de la orientación se ha creado la clase **IMUfilter**, que mediante su método principal fusiona la entrada proporcionada por los acelerómetros y los giroscopios devolviéndonos el cuaternión que nos permite orientar nuestro objeto virtual.

Ya que no es posible reajustar el *yaw,* se ha implementado la opción de devolver el objeto virtual a su orientación inicial. Si orientamos nuestro Wiimote de igual forma que está orientado el objeto virtual en su estado inicial, haremos corresponder la orientación de ambos pulsando el botón A del Wiimote. El sistema también nos permite ajustar el parámetros *beta* del filtro y comprobar cómo afecta esto a la forma de reajustar la orientación.

6.4 Conclusiones

Al iniciar la aplicación deberemos colocar el Wiimote en la misma posición que el objeto virtual, por defecto con los botones mirando hacia nosotros. Una vez colocado en la posición inicial pulsaremos el botón A del Wiimote. Desde ese momento podremos girar el Wiimote real y el Wiimote virtual reflejará la misma orientación.

A fin de poder demostrar la eficacia del **IMUfilter** se permite manipular el parámetro *beta* del filtro mediante los botones 1 y 2 del Wiimote, que lo aumentan y disminuyen respectivamente. Por defecto el parámetro *beta* será inicializado a 300, ya que con ese valor la orientación se reajusta suavemente y una vez reajustada la vibración es mínima.

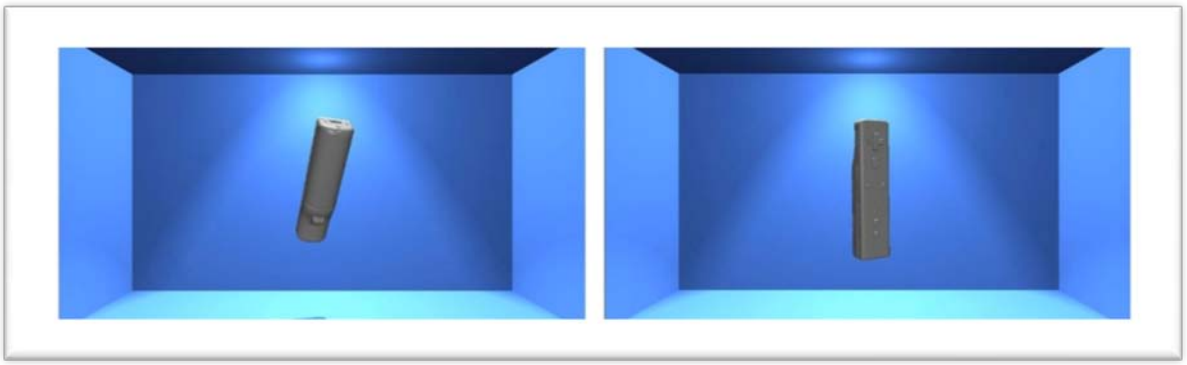

Figura 30. Comparativa entre no usar el filtro (izquierda) y si usarlo (derecha)

En la figura superior (Figura 30) se puede apreciar el efecto de aplicar el filtro y no aplicarlo. Para ello se parte de la orientación inicial, se realizan varios giros aleatorios y a distintas velocidades en los 3 ejes del Wiimote durante 10 segundos y finalmente se retorna a la posición inicial. En la imagen de la izquierda se aprecia la posición del Wiimote virtual con *beta* igual a 0, es decir, sin filtro, tras los 10 segundos y retornar con el Wiimote real a la posición inicial. En la imagen de la derecha podemos ver el resultado del mismo proceso, pero con un valor de *beta* igual a 300. Se puede comprobar cómo en la ejecución con el filtro aplicado el *pitch* y el *roll* son reajustados correctamente, demostrándose el correcto funcionamiento del filtro. Sin embargo también es apreciable como el *yaw* no es reajustado, ya que no es posible con los sensores de inercia del Wiimote.

A modo de cierre de estas conclusiones se muestra una serie de fotografías (Figuras 31, 32, 33, 34 y 35) que reflejan como el Wiimote virtual adopta la misma orientación que el Wiimote real.

Figura 31 Test de orientación 1

Figura 32 Test de orientación 2

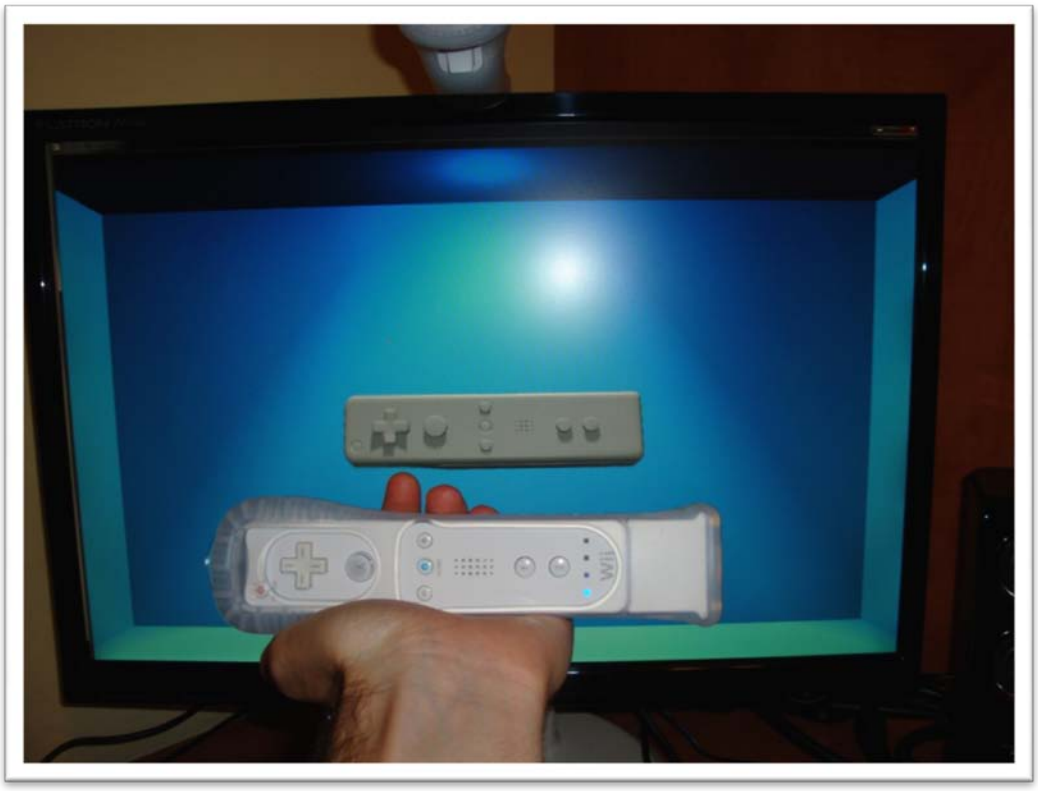

Figura 33 Test de orientación 3

Figura 34 Test de orientación 4

Figura 35 Test de orientación 5

7. PlayStation Eye y OpenCV: obtener la posición 3D de una esfera iluminada.

7.1 Análisis

El objetivo de esta segunda parte de la utilidad para obtener un *motion controller* con 6 grados de libertad era obtener la posición del objeto. Para ello se tomó como ejemplo el PlayStation Move. La posición de éste es obtenida gracias a la combinación de la cámara PlayStation Eye y la esfera iluminada ubicada en el extremo de éste. El uso de una esfera en lugar de cualquier otra forma es porque la forma esférica se observa igual independientemente de la orientación. Además tan solo es necesario obtener el centro y el radio de la esfera para determinar la posición y distancia del objeto.

OpenCV fue la librería elegida. Ésta fue elegida por su sencillez, la multitud de operaciones que implementa y por su amplia documentación. A continuación se describirá brevemente la librería y algunas de sus funcionalidades.

7.1.1 OpenCV

OpenCV es una librería libre diseñada para proporcionar estructuras de datos y métodos para las distintas aplicaciones enmarcadas dentro del campo de la visión por ordenador.

Algunas de las áreas para las que OpenCV proporciona soporte son:

- Estimación del movimiento de la cámara.
- Filtrado de imágenes.
- Reconocimiento facial.
- Reconocimiento de gestos.
- Interfaces hombre-máquina.
- Robots móviles.
- Identificación de objetos.
- Segmentación.
- Estereopsis (percepción de profundidad usando 2 cámaras).

OpenCV fue inicialmente desarrollada por Intel y su licencia es BSD, por lo que puede ser usada tanto en aplicaciones con propósito comercial como en investigación. La primera versión alpha de OpenCV fue presentada en la IEEE Conference on Computer Vision and Pattern Recognition en el año 2000. Entre los años 2001 y 2005 se desarrollaron varias versiones beta, siendo finalmente lanzada la versión 1.0 en 2006. En 2008 OpenCV recibió el

apoyo de Willow Garage, una empresa de robótica. Desde entonces la librería ha ido mejorando en prestaciones y rendimiento.

OpenCV está principalmente escrita en C para facilitar la portabilidad, sin embargo desde la versión 2.0 se incorporan muchos métodos y estructuras de datos escritas en C++, siendo más difíciles de portar. Para este proyecto se ha utilizado la versión 2.1 de OpenCV. Esta versión, al igual que la 2.0 presenta un rendimiento superior con respecto a las versiones 1.x, ya que muchos métodos han sido reimplementados en C++. Sin embargo las versiones en C de los métodos siguen estando accesibles.

7.1.2 Primera aproximación: cámara web y naranja

Dado que en un principio no se disponía de una cámara PlayStation Eye ni de una esfera iluminada se decidió comprobar si era viable obtener resultados admisibles sin estos dos dispositivos. En substitución de éstos se empleó una webcam y una naranja. Mediante OpenCV se procesó cada imagen capturada por la cámara con el objetivo de encontrar la naranja. El algoritmo implementado y aplicado a cada *frame* fue el siguiente:

- Aplicar un filtrado por color para generar una máscara donde todo lo que no fuera de color naranja fuera negro.
- Aplicar la máscara sobre el *frame* en escala de grises (un solo canal).
- Aplicar un filtrado morfológico: erosionar para eliminar el ruido y dilatar para compensar la erosión en elementos distintos del ruido y no ocultar los contornos de la naranja.
- Aplicar un suavizado para facilitar la detección de contornos.
- Usar el método HoughCircles [11] que convierte la imagen en una imagen de contornos y a continuación detecta círculos en ésta. Al aplicar la máscara limitamos el área de búsqueda del *HoughCircles*, evitando así detectar otros círculos.

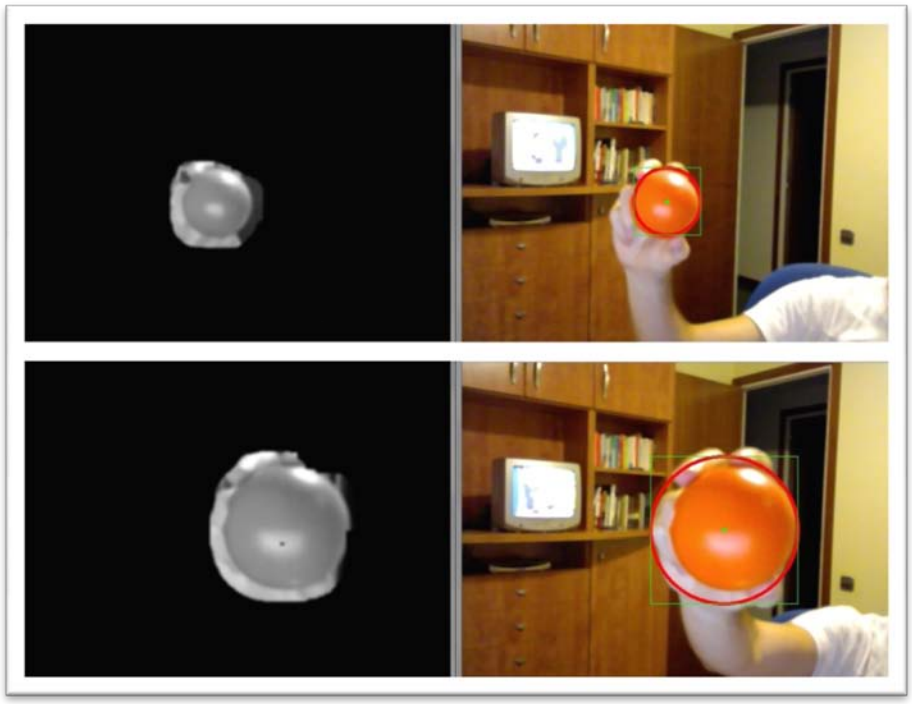

Figura 36 Detección de naranja con cámara web y *HoughCircles* de OpenCV. Las imágenes de la izquierda son el resultado de aplicar la máscara sobre la imagen de grises

El algoritmo funciona correctamente (Figura 36), es decir, detecta la naranja, sin embargo los resultados no son admisibles por varios motivos. En primer lugar la detección no se produce satisfactoriamente en todos los *frames* procesados. Si la naranja se traslada lentamente la detección es más frecuente que cuando se traslada a más velocidad. Esto se produce por la baja tasa de imágenes por segundo que puede procesar la cámara, 30, lo habitual en cámaras web domésticas. Pero también es debido a la iluminación que afecta a la naranja. Según como incide la luz, la naranja presenta distintas tonalidades, sombreados y reflexiones que impiden detectar correctamente los contornos de la fruta. Provocando unas veces que no se detecte la naranja y otras que el centro y el radio de esta sean imprecisos como en la segunda imagen de la figura 36.

Como los resultados obtenidos con el filtrado de color y HoughCircles no fueron admisibles se testeó otro algoritmo. En esta ocasión no se implementó, ya que venía como ejemplo en el SDK de OpenCV. El algoritmo en cuestión es CamShift (Continously Adaptative Mean Shift) [12] y se fundamenta en buscar similitudes en la imagen con las definidas en un histograma de colores. Para definir dicho histograma el programa nos da la opción de marcar una zona de la imagen. Marcando la naranja se obtiene un histograma de tonos anaranjados y la región que contiene más similitud con el histograma es marcada, es decir, nuestra naranja. El algoritmo también funciona correctamente, pero presenta varios inconvenientes:

- Al desplazar el objeto rápidamente el algoritmo lo pierde y se queda buscando en la zona donde lo perdió, requiriendo volver a pasar por la zona con el objeto para que sea detectado de nuevo.

- Al tratarse de únicamente un filtro por color, se desconoce la morfología del objeto y por lo tanto es complicado determinar el radio o el centro de la naranja.
  CamShift es descartado pero nos permite comprender la necesidad de no realizar únicamente un filtrado de color, sino también de emplear un método que base la detección en la morfología del objeto, como hacemos en nuestro algoritmo. Con HoughCircles se consiguen detecciones del centro y el radio muy acertadas incluso con la naranja parcialmente ocluida (Figura 37).

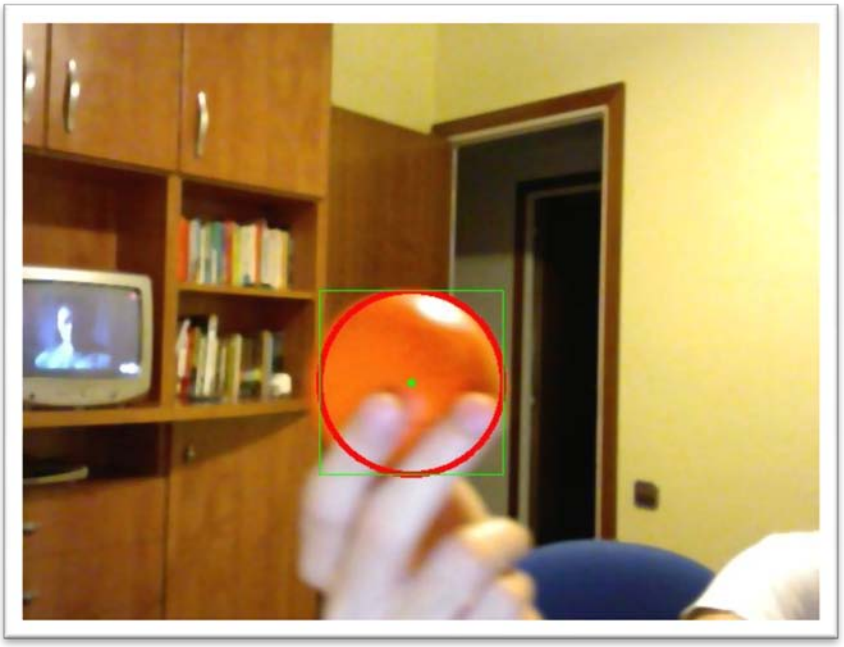

Figura 37 HoughCircles con naranja parcialmente ocluida

Aceptando como válidos los dos fundamentos de nuestro algoritmo, el filtrado por color y el filtrado morfológico, decidimos descartar el uso de una cámara web y una naranja por todos los inconvenientes no salvables que estos presentan. La adquisición de una cámara PlayStation Eye y una esfera luminosa resultan imperativas para obtener resultados admisibles haciendo uso de nuestro algoritmo.

7.1.3 Librería CL-Eye

Para usar la PlayStation Eye en Windows se instalaron los drivers CL-Eye. Gracias a éstos es posible configurar la cámara en todos los modos que soporta, siendo 75 *fps* la tasa máxima de imágenes a 640x480, y 125 *fps* la tasa máxima a 320x240. Aunque de este modo

OpenCV detecta la cámara, las opciones de configuración de ésta son limitadas. Por este motivo se instaló el CL-Eye SDK, que contiene la librería CLEyeMulticam. Con esta librería podemos interactuar igualmente con OpenCV, pero ganamos en gestión de la cámara.

7.1.4 Esfera luminosa

El uso de una esfera iluminada proporciona una detección más robusta con independencia de las condiciones de iluminación del entorno. Al estar iluminada el color se muestra de forma uniforme, sin sombreados ni cambios de tonalidad ni reflejos.

Es posible adquirir una esfera iluminada (Figura 38) en varias tiendas online e incluso en tiendas físicas de electrónica. Los precios oscilan entre los 3 y 15 euros para los modelos de 6 cm, el tamaño que más se aproxima al ancho del Wiimote. Estas esferas las venden para decoración o para hacer *juggling* (malabares). Las primeras son más económicas pero más sensibles a caídas y golpes y las segundas son más caras, pero resisten caídas y golpes e incluso botan. Para la aplicación nos decantamos por el modelo más económico, el cual tuvimos que modificar. Éste incluía en su interior 3 leds que se iban encendiendo y apagando alternativamente. Dichos leds fueron remplazados por un único led azul de más intensidad.

La intensidad lumínica resulta un factor importante ya que con las pilas con baja carga o empleando leds de poca intensidad se obtienen peores resultados.

Figura 38 Esfera iluminada desmontada

7.2 Diseño

Reafirmada la necesidad de usar una PlayStation Eye y una esfera luminosa, únicamente deberemos definir el algoritmo para la detección de la esfera. En los siguientes puntos se detallará el proceso de diseño del algoritmo final. Para ello iremos tratando el algoritmo partiendo de las primeras versiones funcionales y explicando las modificaciones e incorporaciones que éste fue sufriendo en pro de una mejor detección.

Remarcar que todo el testeo del algoritmo se ha realizado en un único entorno. Sin embargo se ha intentado aprovechar al máximo dicho entorno enfocando la cámara a distintos lugares de la habitación y comprobando el correcto funcionamiento del algoritmo en distintas condiciones de iluminación.

7.2.1 Detección de esfera: versión básica

La primera versión del método para detectar la esfera luminosa consistió en portar directamente la implementación de la versión para detectar la naranja. Para poder modificar los parámetros del algoritmo en tiempo real y comprobar su efectividad, se crearon dos ventanas (Figura 39). En la primera ventana mostramos las imágenes RAW en color capturadas por la cámara. En esa ventana también se recuadra la esfera cuando es detectada y se cruzan una línea horizontal y otra vertical a lo largo del ancho y alto de la imagen respectivamente, que indican mediante su intersección el centro de la esfera. En esta primera ventana también se incluyen dos *trackbars*. Cada *trackbar* nos permite ajustar el valor del parámetro al que está asociado dentro del rango definido. Una *trackbar* nos servirá para ajustar la ganancia y la otra la exposición, siendo una configuración recomendable máxima exposición y mínima ganancia. En la segunda ventana se muestra la máscara obtenida al aplicar el filtrado por color y el filtrado morfológico (erosión y dilatación). También se muestran varias *trackbars* para configurar los parámetros de los filtros.

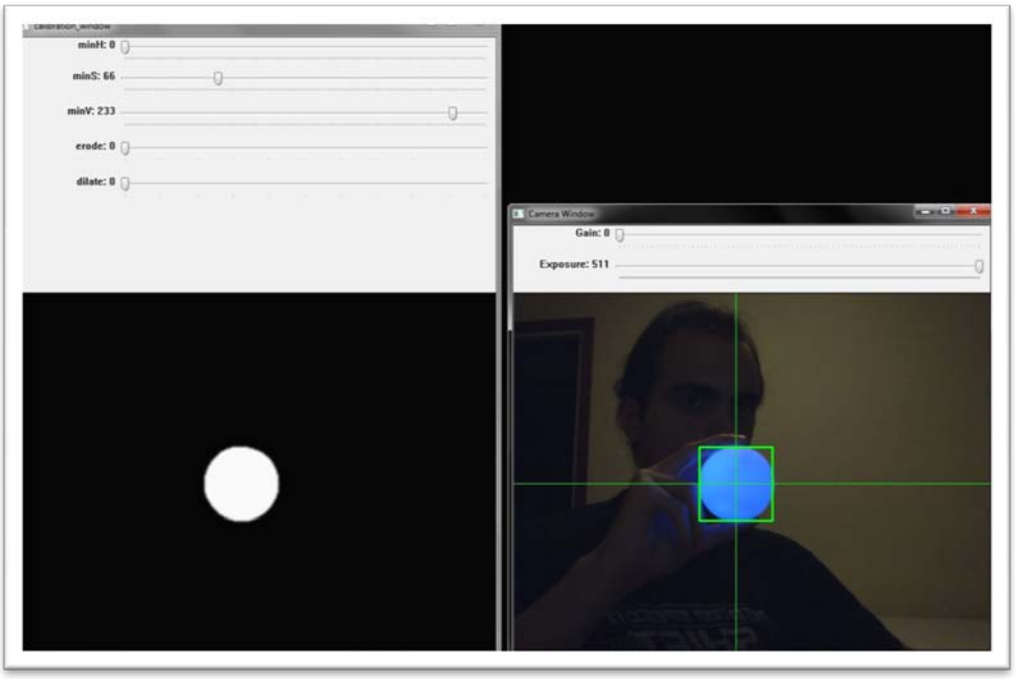

Figura 39 Ventanas de OpenCV con respectivas *trackbars*. Máscara (izquierda). Imagen RAW (derecha).

A continuación se recordará el algoritmo empleado y se detallará cada paso de éste:

1. Filtrado por color.
2. Filtrado morfológico.
3. Suavizado.
4. Houghcircles.

Es importante remarcar que el algoritmo toma como entrada la imagen RAW y el resultado de aplicar el filtro por color sobre ésta es guardado en una imagen que llamaremos imagen máscara. Esta imagen máscara es la que actuara como entrada y salida de los demás procesos del algoritmo.

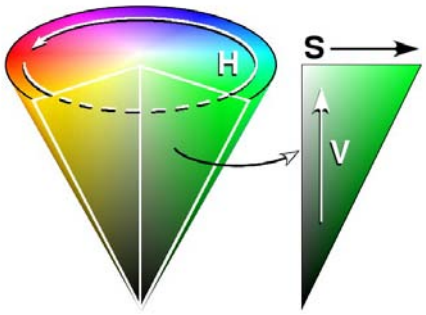

Figura 40 Representación gráfica del espacio de color HSV

Para realizar el filtrado por color se convierte la imagen BGR a HSV (tono, saturación, brillo). En OpenCV las imágenes están por defecto en el espacio de color BGR, que es lo mismo que el espacio RGB (rojo, verde, azul), pero con los canales cambiados de orden. Convertir la imagen a HSV (Figura 40) nos permite ajustar los parámetros del filtro de color de una forma más intuitiva. Por ejemplo para crear una máscara en RGB o BGR que anule el amarillo deberemos ajustar los 3 canales de la imagen, sin embargo en HSV únicamente manipulando el *Hue* (tonalidad) conseguimos nuestro objetivo (Figura 41).

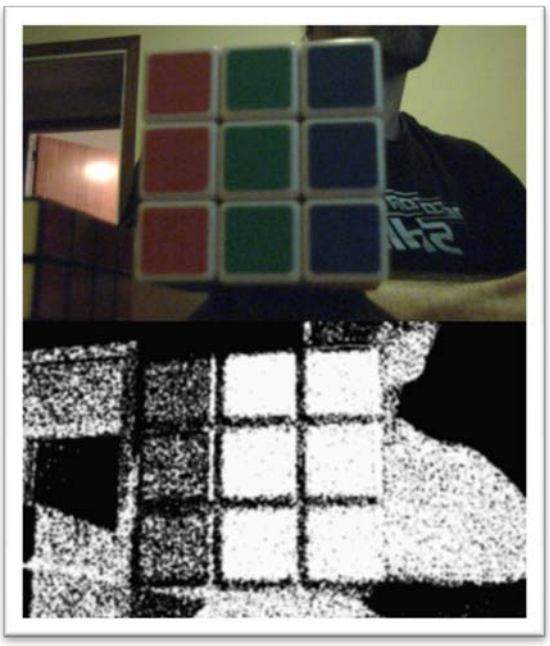

Figura 41 Máscara para quitar amarillo en HSV

Para esta primera versión del filtrado por color únicamente se emplearon 3 parámetros, cada uno relativo al umbral de cada canal. Es decir, para cada canal se modificaba el umbral mediante la *trackbar*. Modificar el umbral quiere decir que se descartan todos los valores por debajo del valor que marca el umbral, que son pintados en negro, mientras que los que están por encima son pintados en blanco. En el ejemplo de la figura 41 no descartamos únicamente el amarillo, sino que se descartan también todos los colores con una tonalidad inferior a la del amarillo. Esto llevo a plantear una mejora que será detallada en la siguiente versión del algoritmo en el punto 7.2.2.

El filtrado morfológico se compone de dos operaciones. Una erosión y una dilatación. Cuando éstas son aplicadas en este orden el proceso recibe el nombre de apertura y si son aplicadas en el orden contrario se denomina clausura.

- Erosión: un píxel tomará el valor 1 en la imagen procesada si ese mismo píxel y todos sus vecinos valían 1 en la imagen original.
- Dilatación consiste en: un píxel tomará el valor 1 en la imagen procesada si ese mismo píxel o cualquiera de sus vecinos valían 1 en la imagen original

Tanto la erosión como la dilatación se definen mediante dos parámetros. El primero es la máscara o *kernel* que se utiliza y el segundo el número de iteraciones o pasadas del filtro. Con el *kernel* se define los píxeles vecinos. Por defecto se emplea un *kernel* de 3x3, esto quiere decir que centrándolo sobre un píxel define como vecinos únicamente a los píxeles contiguos a éste. Para ambas operaciones se fijo el *kernel* por defecto. En cuanto al número de iteraciones se decidió hacerlas configurables, limitando en ambos casos a 10 el número máximo de iteraciones.

El suavizado consistió en aplicar un filtro Gaussiano a la imagen. El filtro se define mediante 3 parámetros. El *kernel* o máscara y las desviaciones estándar en *x* e *y.* Sin aplicar el suavizado HoughCircles es incapaz de detectar un solo círculo. Los parámetros del filtro Gaussiano fueron definidos de forma fija, siendo estos los recomendados por la documentación de OpenCV para poder aplicar satisfactoriamente HoughCircles. Un *kernel* de 7x7 y desviaciones estándar de 1,5.

Tanto el filtrado morfológico como el suavizado no sufrieron más cambios. Las posteriores versiones del algoritmo se centran en mejorar el filtrado por color y sobretodo hacer más precisa la detección de la esfera mediante HoughCircles y otros métodos.

En esta primera versión se emplea únicamente HoughCircles para detectar un círculo en la imagen. Los parámetros son fijados a los recomendados por la documentación de OpenCV. A pesar de que el filtrado de color deja pasar otros elementos además de la esfera, se consigue detectar ésta de forma bastante precisa. La precisión del centro es mucho mayor que la del radio, que oscila unos 7 píxeles, frente a los 2 o 3 del centro. El radio determina la profundidad del objeto virtual, una oscilación así provoca que el objeto virtual avance y retroceda constantemente. Mientras que la variación del centro genera un temblor casi imperceptible del objeto virtual.

Dados los resultados obtenidos mediante HoughCircles, en las siguientes versiones del algoritmo nos centramos en estimar el radio de una forma menos ruidosa pero sin desprendernos de HoughCircles totalmente, ya que es un método robusto en caso de ruido en la imagen o oclusiones parciales de la esfera.

7.2.2 Detección de esfera: mejora del filtrado de color

Para hacer más precisa la detección del color de la esfera se mejoró el método anterior añadiendo más precisión para determinar que colores filtrar. Para ello se amplió el método del umbral para definir un rango. De este modo cada canal se configura mediante dos parámetros, un rango inferior (el umbral) y un rango superior. De este modo se consigue filtrar un mayor número de distracciones (Figura 42). Elementos que emiten luz de tonos azulados e intensos, como un televisor, no se consiguen filtrar mediante el filtro de color, sin embargo el filtro morfológico y el suavizado consiguen minimizar su tamaño en la imagen máscara.

Esta versión del filtro de color se consideró admisible y no sufrió más modificaciones.

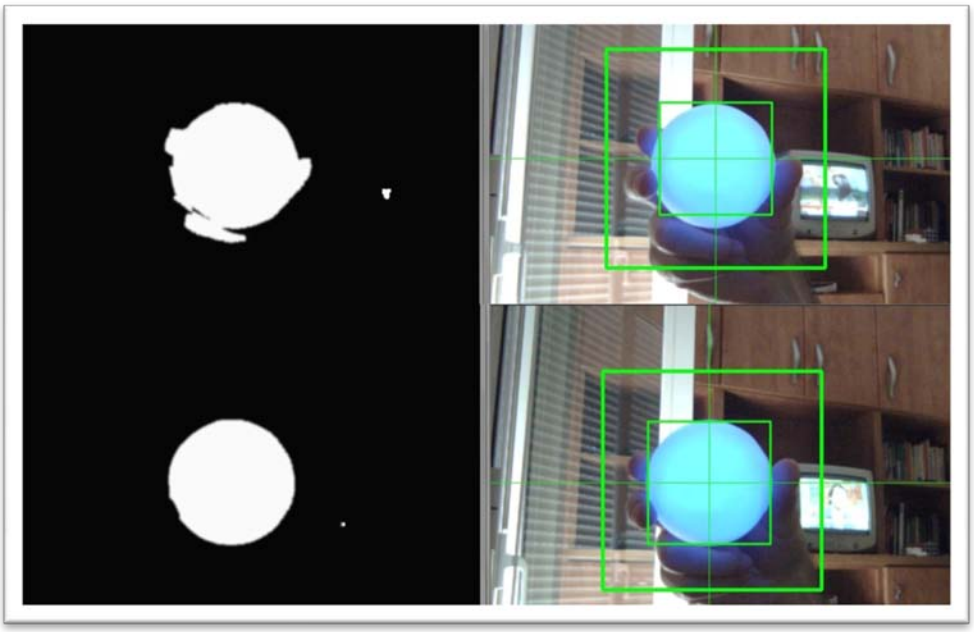

Figura 42 Filtrado anterior (arriba) comparado con el nuevo filtrado (abajo). En la nueva versión limitar el valor máximo del *hue* nos permite obtener mayor precisión

7.2.3 Detección de esfera: mejora de la estimación del radio

Con un filtrado de color perfeccionado se consideró viable implementar un método para estimar el radio de una forma más precisa. Éste consiste en lo siguiente. Primero se detecta el centro y el radio con HoughCircles. Conocido el centro iniciamos una exploración desde éste en las cuatro direcciones. Esto nos da 4 radios con valores muy similares. Promediamos los 4 radios, obteniendo un nuevo radio más preciso que el que nos proporciona HoughCircles y sobretodo menos ruidoso.

Viendo la gran exactitud que nos proporciona este método en la estimación del radio decidimos aprovechar los 4 radios definidos para encontrar un centro más preciso y estable. Para ello se define un cuadrado mediante con los 4 radios obtenidos y se calcula el píxel central de éste, obteniendo de este modo el nuevo centro (Figura 43).

Sin embargo esta forma de estimar el radio y el centro son mucho más sensible que HoughCircles a oclusiones parciales de la esfera. En realidad las oclusiones parciales invalidan totalmente las mediciones realizadas por este método. Por este motivo se decide tomar como válidos los nuevos valores del centro y el radio únicamente cuando la oclusión es mínima. Para ello se comparan los valores del nuevo radio y el nuevo centro con los valores que nos proporciona HoughCircles. Si los nuevos valores son muy dispares en relación a los originales (los proporcionados por HoughCircles), se descartan los nuevos valores y se toman como válidos los valores originales. Esta solución provoca pequeños saltos cuando en un *frame* se dibuja el objeto virtual mediante los valores originales y en el siguiente con los nuevos y

viceversa. Para solucionar este problema se decide aplicar un filtro para suavizar este efecto. El filtro consiste en una interpolación lineal entre el valor actual y los tres valores anteriores.

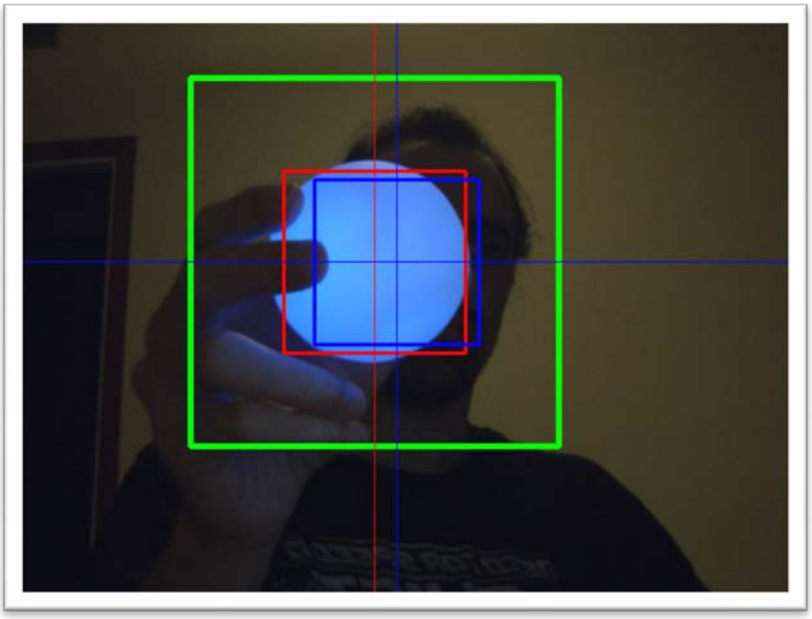

Figura 43 Ejemplo de fallo del nuevo método (azul) por oclusión parcial. En rojo el método por HoughCircles

Como se aprecia en la figura 43 una oclusión parcial hace que el centro y radio calculados por el nuevo método (azul) no sean correctos, mientras que los calculados por HoughCircles sean igualmente acertados. En estas situaciones se elegirán los valores originales y serán interpolados con los obtenidos en los 3 *frames* anteriores.

Esta versión de la detección del radio y el centro de la esfera se considera admisible y se convierte en la versión definitiva de la utilidad.

7.2.4 Detección de esfera: mejora del rendimiento

Aunque en el equipo utilizado el algoritmo realiza la detección a 60 imágenes por segundo a 640x480 píxeles y a 125 imágenes por segundo a 320x240, se realizó una optimización para poder ejecutar la utilidad a 60 imágenes por segundo en equipos menos potentes. La optimización consiste en limitar el área de búsqueda de HoughCircles en base al centro de la esfera calculado en el *frame* anterior. Conocido el centro en el *frame* anterior, se define un área de interés de 300x300 píxeles si procesamos imágenes de 640x480 y de 150x150 si procesamos imágenes de 320x240, lo que supone casi una cuarta parte del área original. En caso de que en el *frame* anterior no se haya encontrado la esfera se realiza la búsqueda en toda la imagen.

La mejora se ve reflejada en el equipo utilizado, rebajándose el consumo de la CPU de un 27% a un 22%. El área de interés es representada por el recuadro verde en la figura anterior.

7.3 Implementación: módulos y funcionalidades

La implementación de ésta parte de la utilidad está realizada de tal manera que se ejecuta en un hilo aparte. Esto hace posible utilizar los datos de la posición y el radio de la esfera de forma independiente de los de la orientación proporcionados por la primera parte de la utilidad.

Para la ejecución del hilo se ha creado la clase **CLEyeCameraCapture**, que hace uso de la librería CLEyeMulticam. En esta clase se inicializa la cámara PlayStation Eye, se crean las ventanas de OpenCV con sus *trackbars* y se inicializan los parámetros necesarios para ejecutar el algoritmo que detecta la esfera. Una vez inicializados todos los parámetros se ejecuta el hilo a una frecuencia máxima igual a la tasa de imágenes por segundo indicada al inicializar la cámara PlayStation Eye.

Gracias a las *trackbars* de las ventanas de OpenCV podemos modificar la ganancia y la exposición de la imagen. Éstos dos parámetros afectan a la imagen máscara (la imagen de entrada del algoritmo de detección de la esfera) pero también podemos ajustar la imagen máscara mediante los parámetros que determinan filtrado de color (tono mínimo, tono máximo, saturación mínima, saturación máxima, brillo mínimo y brillo máximo) y los que determinan el filtrado morfológico (iteraciones de erosión, iteraciones de dilatación).

Una última *trackbar* nos permite visualizar la diferencia entre los distintos método de detección de la esfera. El primer método es el HoughCircles normal, el segundo el método explicado en el punto 7.2.3, y el tercero la interpolación de las últimas detecciones.

7.4 Conclusiones

Gracias a las mejoras implementadas en el filtrado por color y en la detección de la posición y el radio de la esfera se consigue renderizar el objeto virtual en la escena sin saltos y de forma suave.

La cámara PlayStation Eye tiene dos modos de resolución y dos modos de ángulo de visión, uno a 56 grados y otro a 75 grados de apertura. La combinación de unos modos con otros determinará el área de uso de la aplicación. En el modo de menor resolución podemos mover la esfera de forma más rápida sin que se pierda la posición de ésta y reflejando un movimiento más suave en el objeto virtual, pero a causa de la menor resolución no podemos alejarnos mucho de la cámara, siendo de 40 cm la distancia máxima con una apertura de 75 grados y de 60 cm con 56 grados de apertura. En el modo de mayor resolución no podemos

mover la esfera de forma tan rápida sin provocar que esta deje de detectarse, pero podemos alejarnos más de la cámara, hasta 90 cm con 75 grados de apertura y hasta 120cm con un ángulo de visión de 56 grados.

Es importante remarcar que la utilidad no calcula la posición de la esfera relativa a la cámara. Es decir no nos indica a cuantos centímetros se encuentra la esfera ni en qué lugar del mundo real. Para ello sería necesario conocer, además de la resolución, el ángulo de visión de la cámara y el tamaño real de la esfera. Esta aproximación no produce ningún problema empleando únicamente la esfera e incluso nos permite añadir un factor de escala que nos permite transformar traslaciones pequeñas en grandes traslaciones en el mundo virtual.

Como curiosidad se hizo una prueba con la esfera luminosa y rebajando a 30 los *fps* de la PlayStation Eye. El resultado fue muy satisfactorio gracias a las peculiaridades de la esfera. Esto abre una vía de trabajo futuro que consistiría en adaptar la implementación a webcams.

A modo de cierre de estas conclusiones se mostrará una serie de fotografías (Figuras 44, 45, 46 y 47) donde se aprecia como la posición de la esfera determina la posición donde se renderiza el objeto.

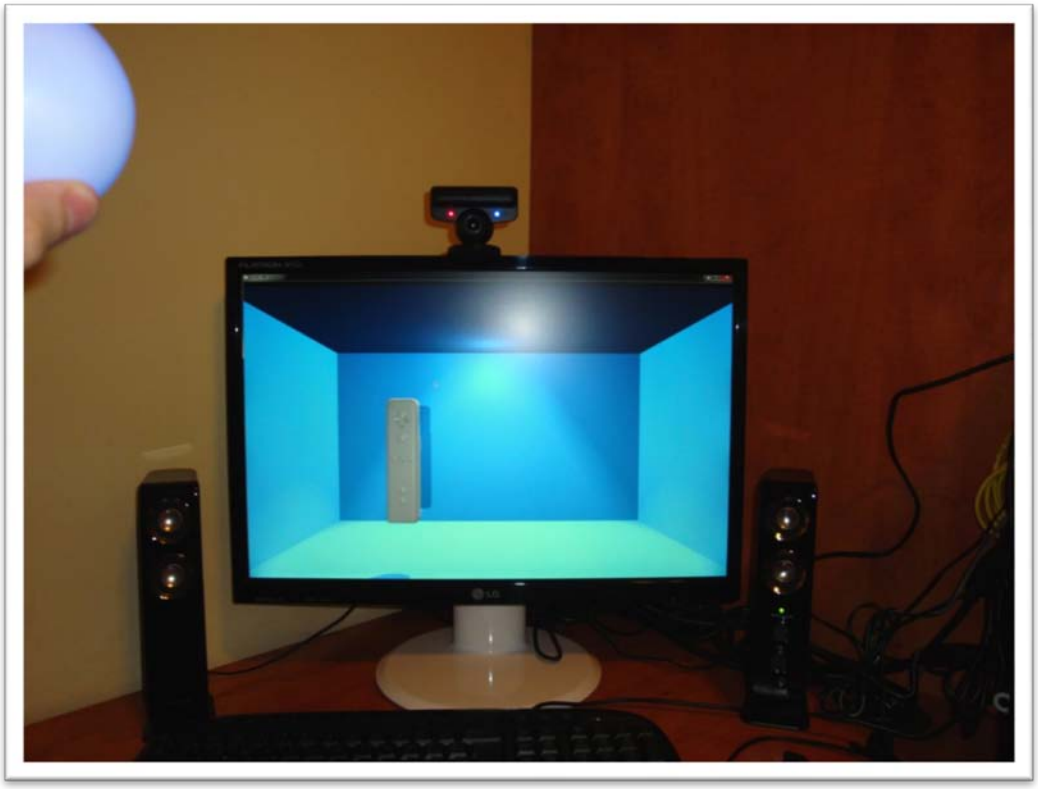

Figura 44 Test de posición 1

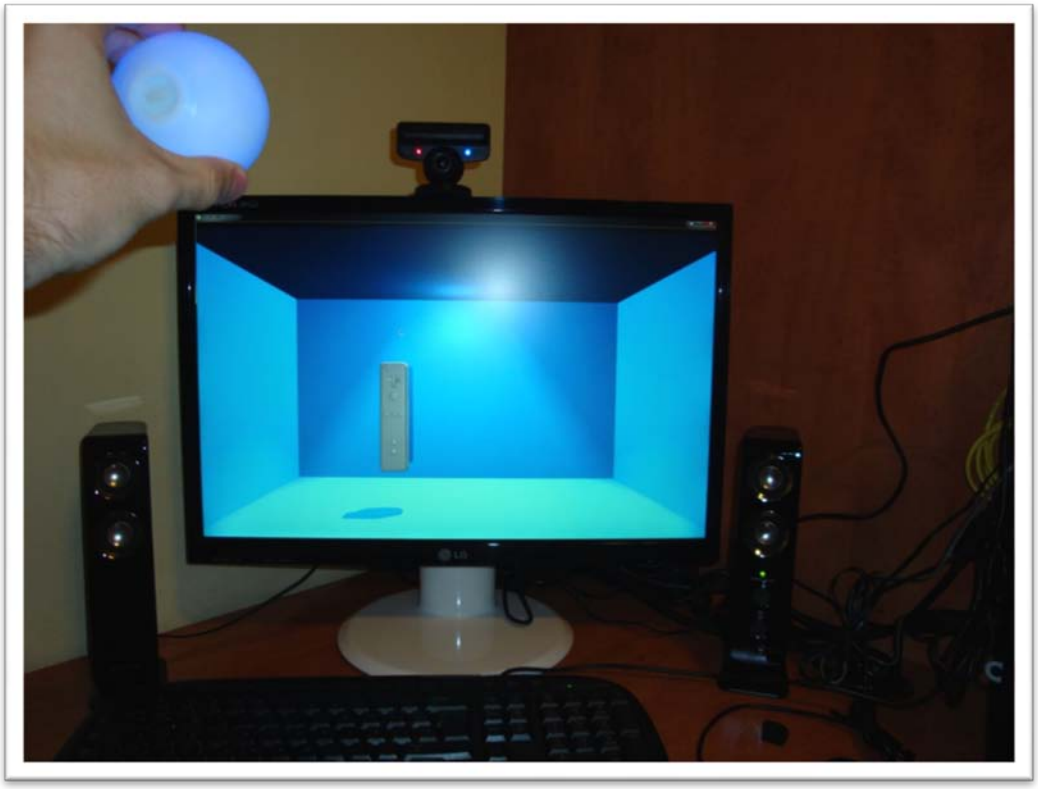

Figura 45 Test de posición 2

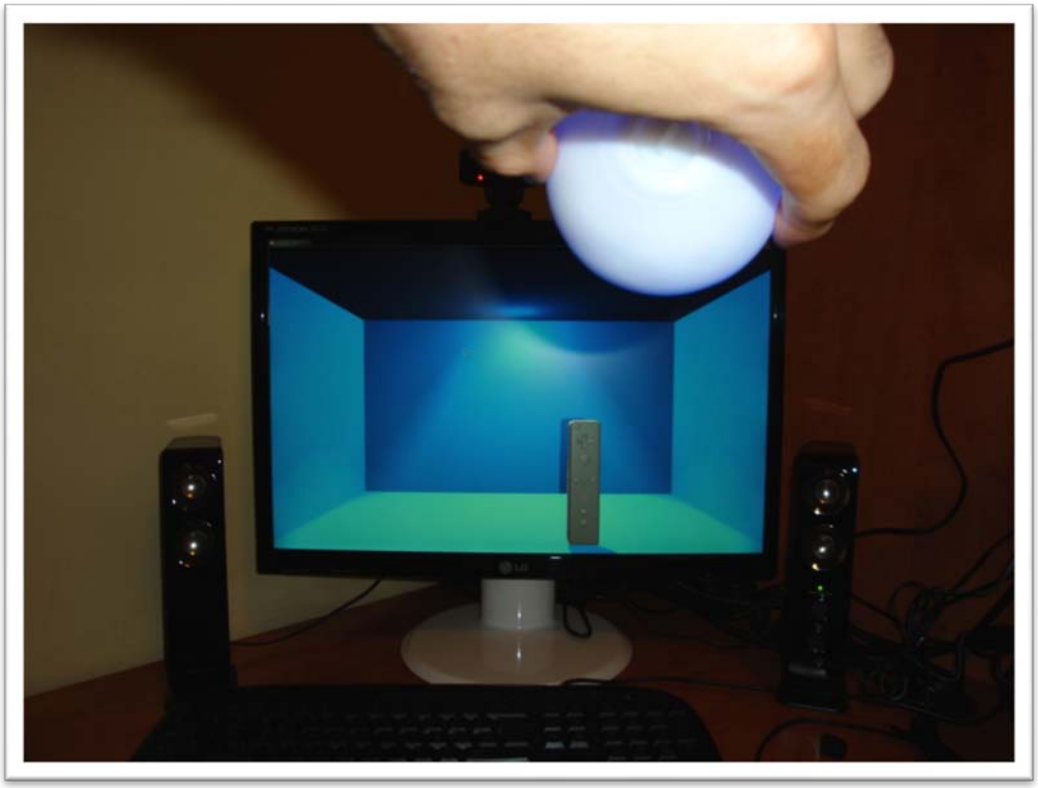

Figura 46 Test de posición 3

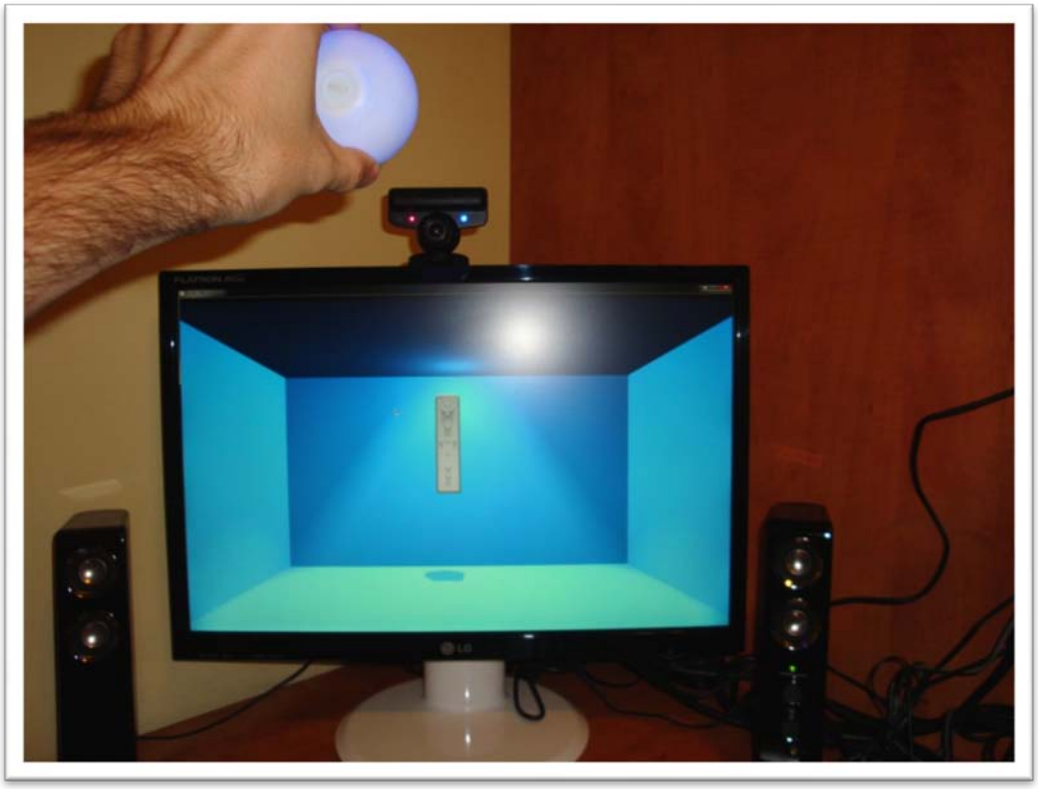

Figura 47 Test de posición 4

8. Motion controller: Integración de la orientación y la posición

Finalizada la implementación de las dos partes de la utilidad, la del cálculo de la orientación mediante el Wiimote y el Motion Plus y la del cálculo de la posición gracias a la cámara PlayStation Eye y la esfera luminosa, se estudió la forma de integrar ambas funcionalidades tanto a nivel de implementación como a nivel físico, es decir, como acoplar la esfera al Wiimote.

8.1 Análisis y diseño de la integración del Wiimote y la esfera iluminada

Un requisito imprescindible en la integración del Wiimote con la esfera era hacerlo de forma que ninguno de los dos elementos resultase dañado y que la unión fuera reversible. La mejor solución que se encontró fue usar una ventosa de doble cara y la funda del Wiimote (Figura 48). La funda dispone de una apertura en su extremo para no tapar la cámara del Wiimote. Aprovechando dicho agujero pasamos una parte de la ventosa por él, quedando la otra parte fuera. De este modo podemos enganchar la esfera luminosa a la ventosa, que proporciona una sujeción bastante firme.

Figura 48 Ventosa de doble cara

Esta es la solución que se empleó, posicionando la esfera en el Wiimote en el mismo lugar que la tiene posicionada el PlayStation Move. Sin embargo ésta configuración presenta un inconveniente. La esfera tapa la cámara, haciendo imposible reajustar el *yaw* del mismo modo que lo haría la Wii. Por este motivo se decidió plantear una solución para reajustar el *yaw* dada la indisposición de la cámara del Wiimote. Esta solución se fundamentó en el uso de la cámara PlayStation Eye.

8.1.1. Reajustando el *yaw* mediante el PlayStation Eye

La solución planteada consistió en pintar una pequeña marca en la esfera ubicada de tal forma que al apuntar con el Wiimote hacía la cámara, de forma paralela al *front* de ésta, la

marca tuviera la misma coordenada horizontal que el centro de la esfera. En esta configuración el *yaw* tomaría el valor cero. Al modificar el *yaw* del Wiimote, la marca se alejaría del centro de la esfera calculado, tomando valores de *x* superiores a los del centro girando en una dirección y valores inferiores girando en la contraria.

La solución parece sencilla, sin embargo no se consiguió implementar. El principal motivo fue el tamaño de la marca. Una marca muy pequeña era eliminada al aplicar la erosión y una marca mayor provocaba falsos positivos con la implementación de HoughCircles de OpenCV. La solución más apropiada sería implementar una versión propia de HoughCircles que subsanase los inconvenientes de la versión de OpenCV, sin embargo una solución así sobrepasa los límites del proyecto.

8.2 Conclusiones

La integración de los dos métodos es satisfactoria. Combinando ambas tecnologías obtenemos un controlador con 6 grados de libertad reales al igual que el PlayStation Move. Se ha conseguido ampliar las capacidades del Wiimote y asemejarlas a las del controlador de Sony. Sin embargo se considera que la solución final es claramente mejorable en pro de parecerse más al tándem PlayStation Move-Eye.

En primer lugar sería requisito indispensable conseguir reajustar el *yaw*, ya sea bien rediseñando el posicionamiento de la esfera de tal modo que no tape la cámara del Wiimote o con algún tipo de marcador que podamos identificar con el PlayStation Eye.

En segundo lugar se debería testear más a fondo el algoritmo para detectar la esfera. Probarlo en diferentes entornos y estudiar formas de hacerlo más dinámico y con menos intervención del usuario en el proceso de calibración.

La esfera también es un factor importante. Ésta no es exactamente igual a la empleada en el PlayStation Move. Las del controlador de Sony además de cambiar de color son de goma blanda. Es decir, que entre los leds del interior de la esfera y el exterior de ésta solo hay una fina capa de goma blanda. Esto hace posible obtener mucha más fuerza lumínica que con nuestra esfera de precio módico, que es completamente rígida y atenúa la luz emitida por el led azul de su interior.

Otro aspecto crucial que haría incrementar la similitud entre nuestra utilidad y el PlayStation Move sería el uso de los acelerómetros para estimar traslaciones cuando la esfera este ocluida o el algoritmo no consiga detectarla. Sin embargo, para estimar la traslación sería un requisito previo indispensable conocer la orientación del Wiimote con exactitud, y dado el problema del *yaw* eso no es posible.

Un pequeño inconveniente al unir la esfera y el Wiimote se produce cuando movemos el Wiimote de una cierta manera. Considerando que el pivote de rotación y traslación del objeto virtual está ubicado en la posición del la esfera, es decir, en el extremo del objeto, al dejar la esfera fija y rotar alrededor de ésta todo funciona perfectamente, sin embargo, si

dejamos fija la parte contraria a la esfera y rotamos el Wiimote la rotación se convierte también en una traslación del objeto virtual. Este fenómeno se produce si el factor de escalado es muy alto, por ello se recomienda no usar factores elevados.

A modo de cierre de estas conclusiones se mostrará una serie de fotografías (Figuras 49, 50, 51 y 52) donde se puede apreciar como la posición y orientación del Wiimote determinan la posición y orientación del objeto virtual.

Figura 49 Test de posición y orientación 1

Figura 50 Test de posición y orientación 2

Figura 51 Test de posición y orientación 3

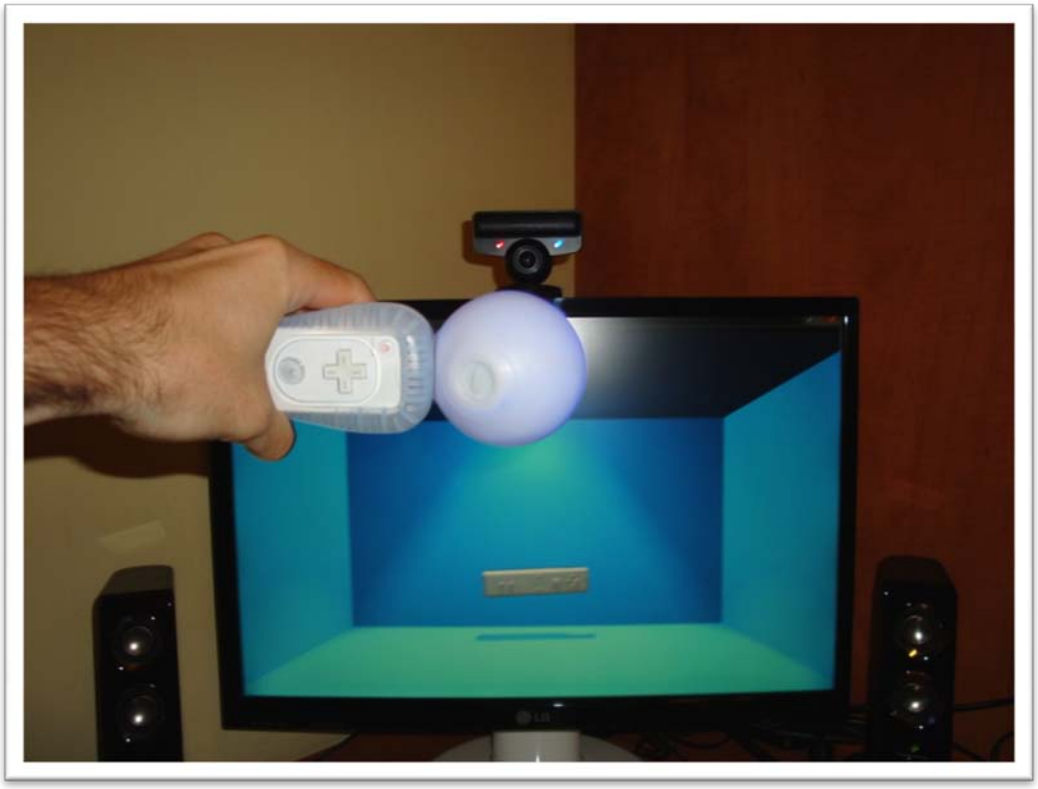

Figura 52 Test de posición y orientación 4

PARTE 3 - WII-IMMERSION

9. Integración de las dos utilidades: Wii-immersion

9.1 Integración de las dos utilidades en el GTI Framework

En este punto se explicará la relación entre los módulos implementados y los componentes del GTI Framework. Para ello presentará un diagrama (Figura 53) que engloba toda la aplicación y se procederá a explicar los componentes del GTI Framework.

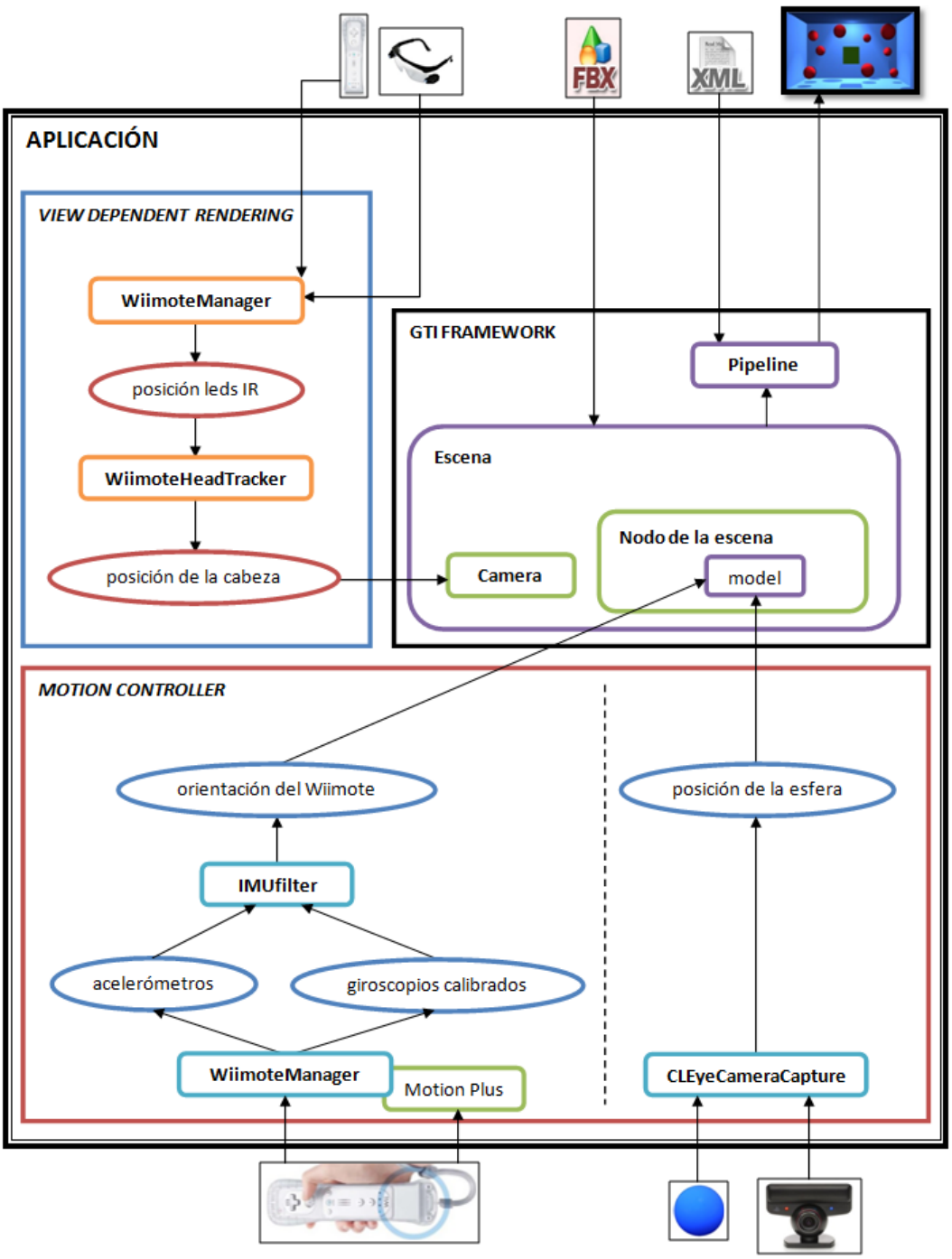

Figura 53 Diagrama de la aplicación

Para visualizar la escena se ha empleado el GTI Framework. El uso de éste permite renderizar una escena de forma muy sencilla. Para ello simplemente se requiere de un archivo con extensión FBX que contenga la escena que queremos visualizar (en nuestro caso la Cornell Box con distintos objetos y luces) y un archivo XML que diga como visualizar la escena, indicando por ejemplo que *shaders* usar y bajo qué condiciones. Dicho archivo XML que define la *pipeline* venía ya incluido en el GTI Framework.

En la aplicación desarrollada se emplean las clases **FBXScene** y **Pipeline** del GTI Framework. Mediante la primera creamos un contenedor donde se almacena la escena importada del archivo FBX y mediante la clase Pipeline ejecutamos la *pipeline* definida en el XML.

La escena importada contiene una cámara y distintos nodos u objetos de la escena, entre ellos nuestro objeto virtual que queremos trasladar y rotar. Para manipular la cámara de la escena se emplea la clase **Camera** del GTI Framework y para manipular el objeto se emplea la clase **SceneNode**.

La relación entre las utilidades implementadas y los componentes del GTI Framework es la siguiente:

- La posición de la cabeza obtenida mediante nuestro *head tracker* nos permite modificar las matrices de vista y proyección de la cámara.
- La posición y orientación obtenidas mediante nuestro *motion control* nos permiten modificar la matriz *model* que define la posición y orientación del objeto que queremos manipular.

9.2 Inicialización de los parámetros mediante XML

Para inicializar la aplicación con los parámetros deseados se ha creado la clase estática **VariablesXML** que nos permite importar los parámetros más relevantes para la inicialización de la aplicación.

Dichos parámetros han sido definidos en el archivo *config.xml* y se listan a continuación:

- Nombre del objeto a manipular de la escena.
- Nombre de la cámara de la escena.
- Valor del parámetro para ajustar el filtro de la orientación.
- Número de iteraciones del calibrador del Motion Plus.
- Valor del escalado del movimiento del *view dependent rendering.*
- Distancia en milímetros entre los dos marcadores.
- Modo de resolución de la cámara.
- Imágenes por segundo de la cámara.

- Tono mínimo.
- Saturación mínima.
- Brillo mínimo.
- Tono máximo.
- Saturación máxima.
- Brillo máximo.
- Número de iteraciones de erosión.
- Número de iteraciones de dilatación.
- Valor de la ganancia.
- Valor de la exposición.
- Factor de escalado en *X* de la traslación del objeto.
- Factor de escalado en *Y* de la traslación del objeto.
- Factor de escalado en *Z* de la traslación del objeto.

9.3 Configuración de los parámetros

Ambas utilidades disponen de varios parámetros modificables en tiempo de ejecución ajustar dichos parámetros permite, entre otras cosas, evaluar los beneficios de las mejoras implementadas en las distintas versiones de las dos utilidades. Los parámetros relativos a OpenCV son configurados mediante las *trackbars* de las ventanas. En cambio los parámetros del *head tracker* y del Motion Plus pueden ser configurados tanto por teclado como con los botones del Wiimote con Motion Plus.

A continuación se listan los distintos parámetros a configurar:

Head tracker

- Incrementar profundidad
  - tecla T o botón UP del Wiimote.

- Decrementar profundidad
  - tecla G o botón DOWN del Wiimote.

- Incrementar el escalado del movimiento
  - tecla Y o botón RIGHT del Wiimote.

- Decrementar el escalado del movimiento
  - tecla H o botón LEFT del Wiimote.

- Centrar la vertical de la cámara
  - tecla espacio o botón B del Wiimote.

- Activar/desactivar *head tracker*
    - tecla F o botones MINUS/PLUS del Wiimote.

Motion controller

- Incrementar valor *beta* del filtro
    - tecla Z o botón 1 del Wiimote.

- Decrementar valor *beta* del filtro
    - tecla X o botón 2 del Wiimote.

- Reajustar la orientación
    - tecla V o botón A del Wiimote.

- Iniciar calibración
    - tecla C.

El resto de parámetros del *motion controller* son gestionados mediante las *trackbars*.

Estos parámetros son:

- tono máximo
- saturación máxima
- brillo máximo
- tono mínimo
- saturación mínimo
- brillo mínimo
- iteraciones de erosión
- iteraciones de dilatación
- ganancia
- exposición

Además se incluye otra *trackbar* para poder visualizar la diferencia entre los distintos métodos de detección de la esfera: HoughCircles, interpolado y preciso.

9.3 Galería final

A continuación se mostrará una serie de fotografías (Figuras 54, 55, 56, 57 y 58) donde se puede comprobar el resultado de la integración de las dos utilidades.

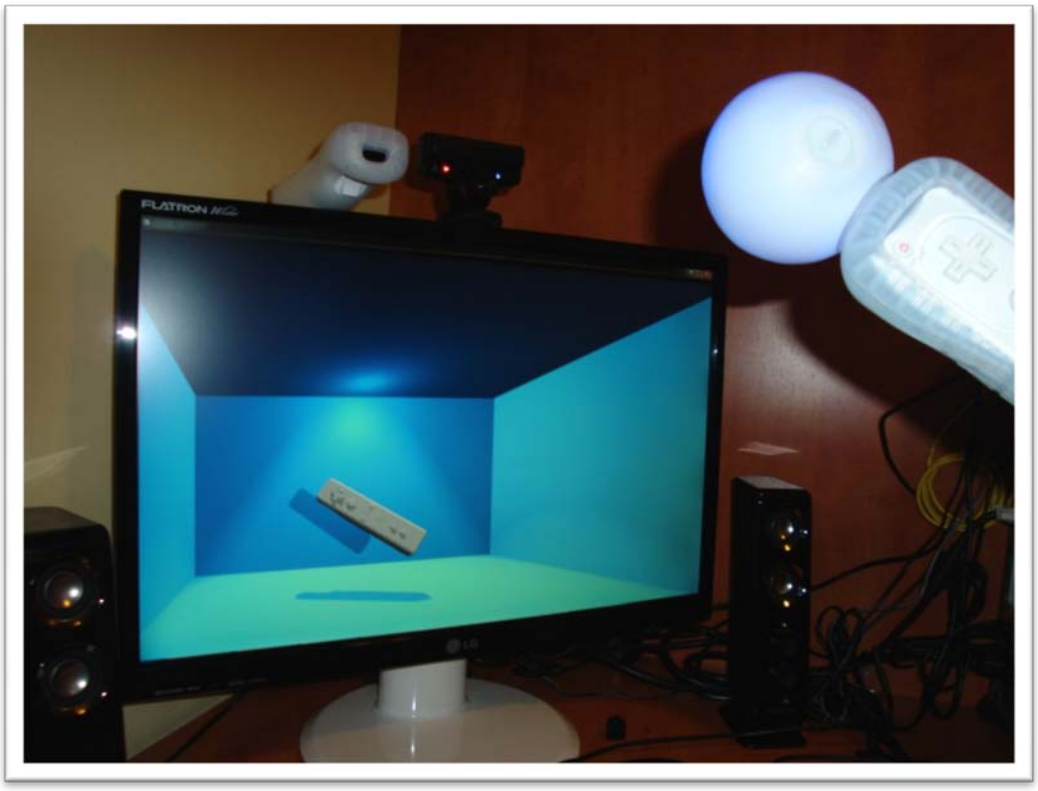

Figura 54 Wii-immersión test 1

Figura 55 Wii-immersión test 2

Figura 56 Wii-immersión test 3

Figura 57 Wii-immersión test 4

Figura 58 Wii-immersión test 5

10. Conclusiones finales

En este último punto se expondrán unas conclusiones generales sobre los resultados obtenidos al implementar i evaluar ambas utilidades.

10.1 Conclusiones generales

A nivel de objetivos asumidos, las dos utilidades hacen lo que se esperaba. La primera nos permite convertir nuestra pantalla en una ventana al mundo virtual visualizado mediante el GTI Framework y la segunda nos permite tocar y manipular ese mundo. De este modo conseguimos una mayor sensación de inmersión ya que ambas utilidades nos hacen más partícipes del entorno virtual y nos permiten una interacción más natural con éste.

Es importante remarcar que el objetivo de éste proyecto era desarrollar ambas utilidades y por lo tanto se ha centrado en dicho proceso de desarrollo. Se han estudiado posibles alternativas y se han realizado varias mejoras a cada utilidad hasta obtener resultados admisibles.

Las utilidades desarrolladas permiten idear distintas aplicaciones que hagan uso de ellas. Por ejemplo con el *view dependent rendering* se podría crear una mascota virtual que viviera dentro de un espacio similar a nuestra Cornell Box y interactuar con ésta mediante nuestro *motion control.*

La posibilidad de usar de manera independiente las dos partes del *motion controller* permite también desarrollar aplicaciones con un menor grado de interacción. Pudiendo usar por ejemplo la esfera luminosa como un puntero 3D o el Wiimote con Motion Plus a modo de volante o para rotar libremente una cámara.

A nivel conceptual, el proyecto permite entender mejor la tecnología implicada en cualquier dispositivo que permite realizar funciones mediante los sensores estudiados. En este aspecto este trabajo define un punto de partida para futuros trabajos basados en hacer uso de las capacidades interactivas de, por ejemplo, el PlayStation Move o del iPhone 4, ya que ambos dispositivos permiten obtener la orientación de forma precisa, y el primero se beneficiaría además del algoritmo implementado para detectar la esfera luminosa.

10.2 Trabajo futuro

En la distintas conclusiones se han tratado distintos puntos a mejorar. Algunos podrían etiquetarse como trabajo futuro.

Sería prioritario implementar una solución para reajustar el *yaw.* Una vez reajustado podríamos estudiar cómo usar los acelerómetros para calcular la traslación del Wiimote en

caso de oclusión de la esfera. Con estas mejoras tendríamos un sistema mucho más similar al PlayStation Move.

Otra posible mejora estaría relacionada con la automatización o semi-automatización de la calibración de la detección de la esfera.

Ajustar el *motion controller* a espacio de cámara en lugar del espacio de mundo. Es decir, que la posición del objeto virtual sea relativa a la de la cámara y que girando la cámara el objeto virtual se posicione dentro de su campo de visión y mantenga su orientación con respecto a la cámara, ampliaría las capacidades interactivas aún más.

Obtener una esfera luminosa de otro color, verde por ejemplo, sería una buena opción que nos permitiría trabajar en un modo para dos usuarios o un modo a dos manos, realizando muy pocos cambios en la implementación actual.

Combinar la visualización del GTI Framework con la imagen real proporcionada por la cámara permitiría trabajar el control gestual aplicado a la realidad aumentada.

En definitiva las dos utilidades, en especial el *motion controller*, permiten desarrollar múltiples aplicaciones con las que interactuar de forma intuitiva.

11. Glosario

Acelerómetro: instrumento para medir la aceleración.

API: Interfaz de programación de aplicaciones. Conjunto de funciones y métodos.

Asíncrono: Que no tiene un intervalo de tiempo constante entre cada evento.

BGR: Rojo, Verde, Azul. Espacio de color.

Bluetooth: Especificación para comunicación inalámbrica.

Clúster: Conjunto de ordenadores para realizar tareas que requieren gran capacidad de computación.

CPU: Unidad Central de Procesamiento. Componente hardware encargado de interpretar y ejecutar las instrucciones.

Cuaterniones: Túpla de 4 elementos ideal para realizar rotaciones.

Desviación estándar: Medida de centralización o dispersión para variables de razón.

Estereopsis: Proceso dentro de la percepción visual que lleva a la sensación de profundidad a partir de dos proyecciones ligeramente diferentes del mundo físico en las retinas de los ojos.

Fps: Tasa de imágenes por segundo.

Frame: Imagen de vídeo.

Framework: Estructura conceptual y tecnológica de soporte, definida normalmente con artefactos o módulos de software concretos.

Frustum: Pirámide de base cuadrada con la punta truncada.

GPU: Unidad de procesamiento gráfico. Tarjeta gráfica. Pieza de hardware encargada de pintar la escena en pantalla.

HoughCircles: Algoritmo para detectar círculos en una imagen a partir de los contornos.

HSV: Espacio de color dividido en tono, saturación y brillo.

Pitch: En aviación, inclinación del morro del avión.

Roll: En aviación, rotación alrededor del eje formado por la cola y el morro del avión.

SDK: Kit de desarrollo de software. Conjunto de herramientas que permite crear aplicaciones.

Síncrono: Que tiene un intervalo de tiempo constante entre cada evento.

Thread: Hilo de ejecución.

Trackbars: Barra de control con un rango limitado para ajustar un parámetro.

XML: Extensible Markup Language. Metalenguaje para etiquetado.

Yaw: Rotación respecto al eje vertical.

12. Índice de figuras

13. Referencias

[1] GTI Framework,
http://galactus.upf.edu/trac/gti-framework/wiki, consultado en Septiembre de 2010

[2] TrackIR
http://www.naturalpoint.com/trackir, consultado en Septiembre de 2010

[3] Head Tracking for Desktop VR Displays using the Wii Remote
http://johnnylee.net/projects/wii, consultado en Septiembre de 2010

[4] OpenCV
http://sourceforge.net/projects/opencvlibrary, consultado en Septiembre de 2010

[5] B. D. Lucas and T. Kanade, *An iterative image registration technique with an application to stereo vision*, 1981.

[6] WiiYourself!
http://wiiyourself.gl.tter.org, consultado en Septiembre de 2010

[7] R. E. Kalman, *A new Approach to Linear Filtering and Prediction Problems,* 1960.

[8] Sem Saelands, *Comparison of Kalman Filter Estimation Approaches for State Space Models with Nonlinear Measurements,* 2005.

[9] S. J. Julier and J.K. Uhlmann, *Unscented filtering and nonlinear estimation, 2004.*

[10] Sebastian O. H. Madgwick, *An efficient orientation filter for inertial and inertial/magnetic sensor arrays,* 2010.

[11] R. O. Duda and P. E. Hart, *Use of the Hough Transformation to Detect Lines and Curves in Pictures,* 1972.

[12] Gary R. Bradski, *Computer Vision Face Tracking For Use in a Perceptual User Interface,* 1998.