Escuela Politécnica Superior de Linares
Grado en Ingeniería de Tecnologías de Telecomunicación

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

DESARROLLO DE UN MECANISMO DE INTERACCIÓN MEDIANTE LA DETECCIÓN DE GESTOS PARA UN ENTORNO 3D.

Alumno: **Mario Beteta Pulido**

Tutor: Raquel Viciana Abad
Depto.: Ingeniería de Telecomunicación

Septiembre, 2016

Escuela Politécnica Superior de Linares

Grado en Ingeniería de Tecnologías de Telecomunicación

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior de Linares

Trabajo Fin de Grado

DESARROLLO DE UN MECANISMO DE INTERACCIÓN MEDIANTE LA DETECCIÓN DE GESTOS PARA UN ENTORNO 3D

Alumno: **Mario Beteta Pulido**

Tutor: Raquel Viciana Abad
Depto.: Ingeniería de Telecomunicación

Fdo. Alumno

Fdo. Tutores

Indice general

1. RESUMEN

Las nuevas tecnologías tienden al desarrollo de interfaces con alto grado de usabilidad. La interacción entre humano y computadora (HCI, Human-Computer Interaction) es un campo de investigación que está en continua evolución. Las empresas tecnológicas en particular, las empresas punteras en el campo de los videojuegos están invirtiendo una parte importante de sus recursos en el desarrollo de nuevos interfaces que seduzcan al usuario con nuevas formas de comunicación con la máquina, aplicables tanto a videojuegos como al control de entornos multimedia. A la hora de hablar de comunicación entre seres humanos los gestos manuales están a la orden del día, y muchos de ellos son entendidos por cualquiera independientemente de su cultura o nacionalidad (e.g. números o direcciones), lo que hace que esta vía de entendimiento resulte más intuitiva y efectiva que otras. Podemos decir por tanto que la utilización de gestos manuales como base de la comunicación usuario-máquina dota al sistema de una gran usabilidad.

En este presente trabajo se ha desarrollado una aplicación para grabar y reconocer gestos, además de integrarla con un entorno 3D. El objetivo principal es analizar las características principales de Leap Motion para el reconocimiento de gestos. Con la ayuda de GRT (Gesture Recognition Toolkit) analizaremos la precisión del reconocedor. Se realizara un estudio valorando los resultados obtenido y su posterior conclusión para integrarlos en un entorno 3D. Las tecnologías usadas son: Lenguaje C++, motor gráfico OGRE, librería GRT, Visual Studio 2012 y Leap Motion.

2. INTRODUCCION

El desarrollo de este trabajo tiene su origen como un Trabajo de Fin de Grado que permite concluir los estudios de Ingeniería de Telecomunicación, cursados en la E.P.S. de Linares de la Universidad de Jaén.

Actualmente hay varios dispositivos de interacción que permiten a las personas interactuar con los ordenadores. Esta interacción se ha ido desarrollando con el

paso del tiempo para mejorar la experiencia de los usuarios intentando hacer que la forma de interactuar con el ordenador sea lo más natural e intuitiva posible.

En el estudio de Gustavo Vidal [2] para obtener los datos de movimiento gestuales existen dos grandes tipos de tecnologías las ópticas y las no ópticas. Están las basadas en la visión u ópticas centradas en la captura de imágenes. Estos sistemas, a través del análisis de la imagen, son capaces de detectar que gesto se ha realizado. Para llevar a cabo la captura del movimiento los sistemas ópticos usan dispositivos de entrada específicos, entre los que cabe destacar:

- ZCam: Cámara que captura imágenes en 3 dimensiones. Utiliza los infrarrojos para capturar la profundidad.
- Stereo cameras: Cámaras que cuentan con dos objetivos sincronizados que capturan la misma imagen, pero desde distinto ángulo.
- Una única cámara.
- De tipo normal: como por ejemplo una webcam o una cámara de video familiar.

Estos dispositivos de captura de imagen son aplicados en distintas tecnologías de análisis de imagen, como las que siguen a continuación:

- Marcas pasivas: Son marcas que se disponen en el objeto a capturar, por ejemplo el brazo. Las marcas reflejan la luz emitida sobre el cuerpo, con el propósito de ser capturada. Luego, a través de cálculos matemáticos y de las imágenes capturadas con las marcas es posible proyectar un cuerpo virtual sobre la imagen.

- Marcas activas: A diferencia de las anteriores, estás emiten luz. Por ejemplo, algunas suelen estar compuestas de LEDS.

- Sin marcadores: Son sistemas de captura de movimiento sin marcas. A través de complejos algoritmos que son capaces de detectar formas

humanas, los cuales parten el cuerpo de las imágenes para poder obtener las capturas de las partes de interés.

También existe otro gran bloque que no se compone de bloques ópticos. Respecto a estos vamos a introducir dos de los más importantes:

- Captura mediante fibra óptica: Se implementan con guantes que están constituidos por fibras ópticas que al doblarse atenúan la luz transmitida pudiendo así controlar la posición de los dedos de la mano. También se puede extrapolar al cuerpo entero.

- Captura mediante ultrasonidos: Se trata de un sistema de emisores que generan ultrasonidos y mediante receptores podemos conocer la posición del emisor.

El dispositivo Leap Motion usado en este trabajo, se puede incluir en el apartado de sistemas ópticos para la detección de movimiento. Funciona mediante tres cámaras infrarrojas localizadas en un pequeño hardware que permiten detectar las manos del usuario y obtener su posición relativa. De esta forma, el usuario es capaz de interactuar con el ordenador simplemente moviendo las manos encima del dispositivo. En los siguientes apartados se explicará más detalladamente.

Comenzar a utilizar una tecnología desconocida como Ogre y Leap Motion siempre entraña cierta dificultad y si le añadimos la barrera del tiempo, aún más. La principal motivación para embarcarme en la realización de este proyecto que une las dos tecnologías es la ausencia de proyectos juntando ambas. También la ausencia de información al respecto en castellano. Hasta ahora sólo se habían publicado pequeños tutoriales en blogs personales pero no existía una plataforma de aprendizaje y documentación equivalente.

3. OBJETIVOS

En este proyecto se pretende analizar las prestaciones del sensor de movimiento de Leap Motion como controlador de gestos para un entorno 3D.

Por lo tanto, es necesaria por un lado la familiarización con la API del dispositivo y el uso de algoritmos como DTW (*Dynamic Time Warping*) para analizar la similitud entre gestos, y por otro lado, el desarrollo de un entorno 3D sencillo que permita incorporar la interacción mediante el uso de este dispositivos en tareas de navegación o selección.

Objetivos básicos:

- Análisis de la API del controlador Leap Motion
- Estudio del algoritmo DTW
- Desarrollo de un sistema de pruebas sencillo que permita analizar la similitud entre gestos.
- Análisis de prestaciones del servicio.
- Implementar las funcionalidades antes descritas en un entorno virtual desarrollado con Ogre3D

4. PRINCIPALES TECNOLOGIAS

4.1 Leap Motion

INTRODUCCION

Leap Motion es un sensor de movimiento, el cual nos permite captar el movimiento realizado con nuestras manos y puede convertirlos en acciones programadas. Su apariencia es la que se muestra en la Figura 4.1.

Figura 4.1: Leap Motion

CARACTERISTICAS TECNICAS

Este primer apartado lo dedicaremos a la parte técnica del dispositivo. Analizaremos todos los componentes como se pueden observar en la figura 4.2, que forman el hardware de Leap Motion, que podemos ver descritos en [3].

Figura 4.2: Partes Leap Motion

Dimensiones

Las dimensiones de este aparato son muy reducidas en comparación con otros tipos de cámaras o reconocedores gestuales, en la figura 4.3 vemos las medidas exactas son 75 mm de largo, 25 mm de ancho y 11 mm de alto.

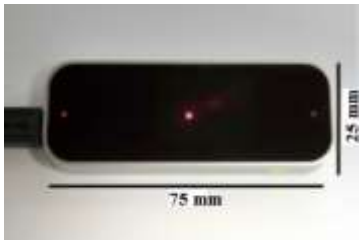


Figura 4.3: Dimensiones Leap Motion

Partes del dispositivo:

Figura 4.4: Camaras Leap Motion

Como podemos apreciar en la Figura 4.4, Leap Motion cuenta con dos cámaras, además de tres LEDS y un microcontrolador. Veamos cada parte detalladamente:

Figura 4.5: Sensor Cámara

Cámaras

En la figura 4.5 observamos la parte más importante del dispositivo, debido a que se ocupa de capturar las imágenes y su correcto funcionamiento condicionará el resto de funciones.

Cada cámara tiene un sensor monocromático, sensible a la luz roja, con una longitud de onda de 850 nm. Este tipo de sensor es capaz de trabajar hasta 200 fps, dependiendo del equipo que estemos usando. Además, cada sensor es de tipo CMOS, que lo utiliza debido a:

- En un sensor CMOS la digitalización de cada pixel se produce dentro de cada celda, con lo cual no es necesario una electrónica exterior. Con esto obtenemos mayor velocidad de captura de imagen y menos espacio para su colocación.

- Estos sensores son más económicos que los sensores CCD.
- Otra de las ventajas es que no produce el fenómeno *blooming*. Ocurre cuando una celda se satura de luz y hace que las demás celdas contiguas también se saturen
- La lectura simultánea de celdas en los CMOS es mayor que en los CCD.
- El consumo eléctrico de los CMOS es menor que el de los CCD.

Iluminación infrarroja

Los LEDs (Indicado en la figura 4.6) se encargan de dilucidar la zona de detección por inundación. Como hemos mencionado anteriormente trabaja con luz infrarroja, que utiliza una longitud de onda de 850 nm. Pueden variar la iluminación, dependiendo de la luz que detecten, para asegurar una misma resolución de imagen.

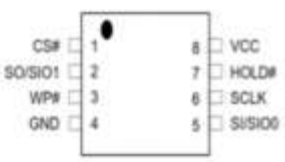
Figura 4.6: Luz infrarroja

Podemos apreciar unas pequeñas barreras de plástico, en detalle en figura 4.7, en cada LED. Esto es debido a que para que sea la iluminación uniforme en toda la zona de cobertura debemos evitar que la iluminación de un LED interfiera en el anterior. También protegemos a los sensores de la saturación de luz.

Figura 4.7: Barrera LED

El microcontrolador

Es un circuito integrado que se suele emplear para hacer la función de BIOS (MXIC MX25L3206E–32M-bit CMOS SERIAL FLASH, descripción en figura 4.8). Se ocupa de controlar el dispositivo, por ejemplo de regular la iluminación, además se encarga de obtener la información de los sensores y enviarla al controlador en el ordenador.



PIN DESCRIPTION

SYMBOL	DESCRIPTION
CS#	Chip Select
SI/SIO0	Serial Data Input (for 1 x I/O)/ Serial Data Input & Output (for Dual Output mode)
SO/SIO1	Serial Data Output (for 1 x I/O)/ Serial Data Output (for Dual Output mode)
SCLK	Clock Input
WP#	Write protection
HOLD#	Hold, to pause the device without deselecting the device
VCC	+ 3.3V Power Supply
GND	Ground

Figura 4.8: Características Microcontrolador

Controlador USB

Leap Motion esta compuesto por un controlador USB mostrado en la figura 4.9. Este controlador es conmumente usado debido a su alta velocidad. Puede soportar USB 3.0.

Figura 4.9: Controlador USB

Envío y recepción de datos

Figura 4.10: Envío y recepción de datos.

Como podemos ver en la imagen 4.10, los datos se envían y se reciben al controlador del ordenador a través de dos puertos serie: UART_RX y UART_TX.

Zona de cobertura

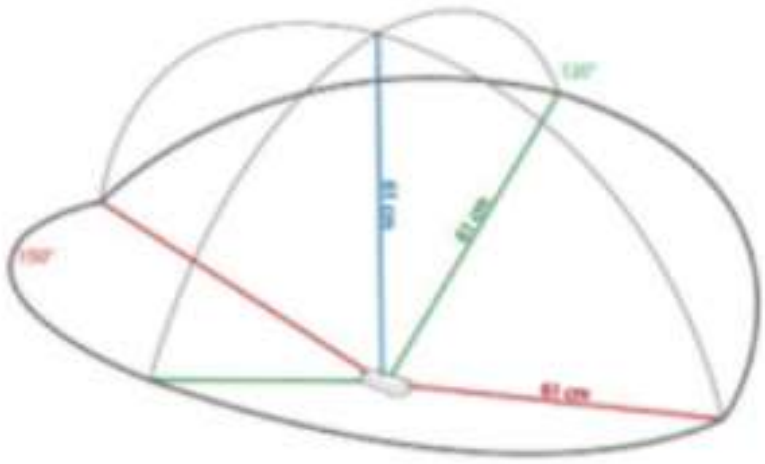


Figura 4.11: Zona de cobertura

En la figura 4.11 observamos la zona de detección del dispositivo. Podemos suponer que se trata de una semiesfera de 61 cm de radio.

Esta zona puede depender de varios factores como son: ángulo de visión de las cámaras y la máxima intensidad que puede entregar el USB. Aunque hay que destacar que el ángulo de visión también le afecta la distancia focal y el tamaño del sensor, como vemos en la siguiente ecuación:

$$\alpha = 2 \times \arctan\left(\frac{d}{2 \times f}\right) \qquad (1)$$

(donde *d* es la diagonal del sensor y *f* la distancia focal)

Tanto el ángulo de visión horizontal de Leap Motion como el vertical son de 150,92º. Estos ángulos delimitan la zona de interacción.

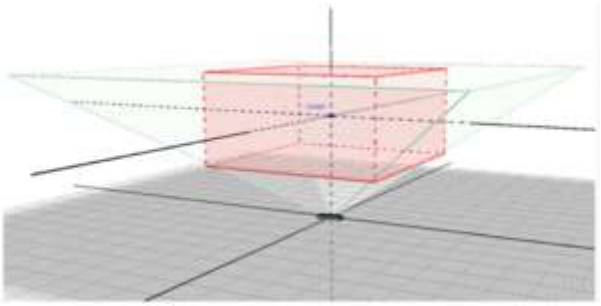


Figura 4.12: Caja de interacción.

En la API del dispositivo se especifica una zona de trabajo llamada "*Interaction Box*" ,como vemos en la figura 4.12,con unas dimensiones de 110.55 x 110.55 x 69.43 mm. Esta zona es la que se detalla en el sistema de coordenadas cartesiano de Leap Motion. Podemos configurar también la altura a la que se encontrara esta altura de interacción. Puede variar entre 7 y 25 cm.

PRINCIPIO DE FUNCIONAMIENTO

El mayor motivo para elegir Leap Motion para nuestro proyecto, es que para identificar los gestos, necesitamos conseguir la mayor información posible del movimiento y posición de las manos. Obtener una información tridimensional es imprescindible para nuestro proyecto.

Para obtener esta información el dispositivo ilumina la zona de detección mediante una luz infrarroja emitida por sus tres LEDs. Esta zona de cobertura tiene las limitaciones anteriormente mencionadas.

Cuando en nuestro caso, las manos, entran en la zona de detección, se produce una reflexión de la luz que llega al dispositivo y la recogen las lentes de las cámaras. Las lentes son de tipo biconvexas, concentran los rayos en el sensor de cada cámara. Los datos obtenidos por los sensores se almacenan en una especie de matriz (imagen

digitalizada), en la memoria del USB, el cual realiza algunos ajustes de resolución mediante el microcontrolador.

Una vez que hayamos ajustado la resolución, los datos son enviados al driver. Estos datos indican un valor de intensidad lumínica de cada pixel de la imagen y se guardan en un buffer. El valor de intensidad lumínica se cuantifica a 8 bits, con lo cual obtenemos 256 posibles valores de luminosidad. El tamaño de la imagen seria de 640 x 120 px.

Las imágenes son analizadas para identificar las manos y los dedos a partir de un modelo matemático de caracterización anatómico. También, obtenemos la profundidad mediante un algoritmo que explicaremos a continuación. Obtenemos la siguiente imagen:

Figura 4.14: Visión Cámaras.

Estas son las dos imágenes que llegan al driver del Leap Motion (se puede acceder a ellas a partir de la versión 2.1 de la API).

Al obtener las imágenes observamos que las lentes producen una distorsión en la imagen óptica, deformando el objeto observado. Vamos a explicar que existen diferentes tipos de distorsión, como podemos observar en la figura 4.15:

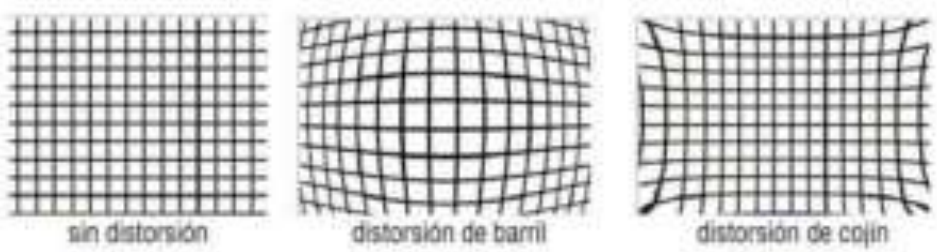


Figura 4.15: Tipos de distorsión.

Como podemos ver en la figura 4.16, en Leap Motion se produce lo que conocemos como distorsión compleja, una mezcla entre distorsión de barril y distorsión de cojín.

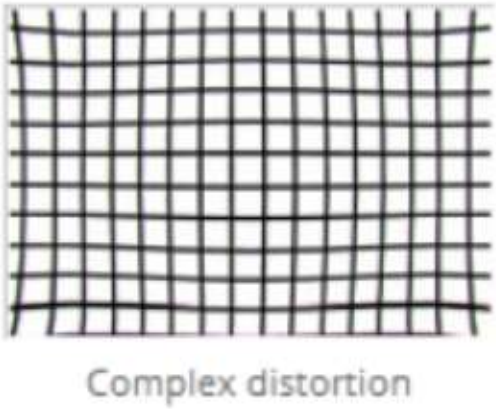


Figura 4.16: Distorsión compleja

Esta distorsión es calibrada por Leap Motion obteniendo un mapa de mallado de puntos (Figura 4.17) de calibrado que se superpone a la imagen captada por el sensor.

Figura 4.17: Imagen mallado.

Por lo tanto, se envían pares de datos de imagen, en los cuales cada imagen va acompañada de su información de distorsión. Esta información es una rejilla de 64 x 64

puntos con dos valores de 32 bits cada uno. Cada valor del mallado define la luminosidad de un pixel de la imagen y se puede obtener los datos de los demás pixeles por interpolación.

Los valores de la cuadricula que caen fuera del rango [0…1] no corresponden a un valor valido de la imagen, por lo tanto se ignoran. Podemos verlo en el siguiente ejemplo:

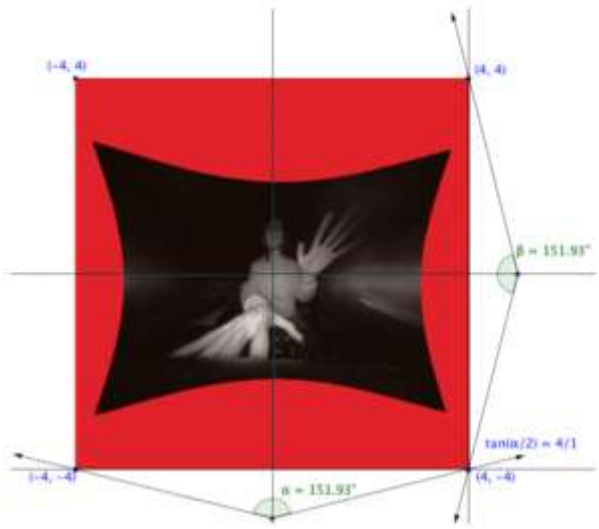


Figura 4.18: Corrección distorsión.

En esta imagen podemos ver la reconstrucción de los datos de una imagen con la distorsión corregida. La imagen se rehace mediante el cálculo de las pistas horizontales y verticales de cada pixel, se puede encontrar el valor del brillo correcto utilizando el mapa de calibración. Las partes rojas de la imagen representan las áreas dentro de la prestación para la que ningún valor de brillo está disponible (el campo de visión real es de menos de 150 grados).

Una vez hayamos obtenido las imágenes correctas y el driver haya identificado las manos y dedos en la zona de detección, podemos determinar la posición de estas en el sistema de coordenadas a través de técnicas de visión estereoscópica.

Básicamente, un sistema estereoscópico funciona del siguiente modo, como nos explica Martín Montalvo en [4]: gracias a la separación de las dos cámaras en el eje X se obtienen dos imágenes con pequeñas diferencias (disparidad).

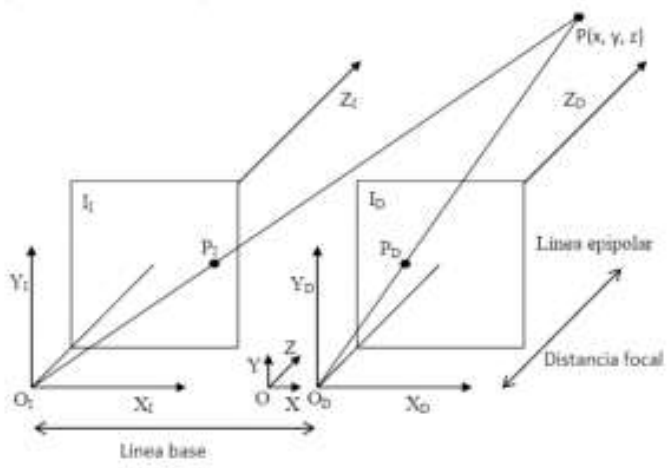


Figura 4.19: Calculo disparidad.

Como se puede observar en la figura 4.19, las dos cámaras —representadas por *Oi* y *Od*— están en el mismo plano *Z*, sobre la línea base. Si trazamos una línea epipolar entre las dos imágenes Ii e Id, dado que *Oi* y *Od* están en el mismo plano *Z* y las dos cámaras tienen la misma distancia focal, podemos ver la proyección del punto *P* en las dos imágenes. Por tanto, se puede obtener un valor de disparidad *d* para cada par de puntos emparejados *Pi(xi,yi)* y *Pd(xd,yd)* dado por d = XI-XD.

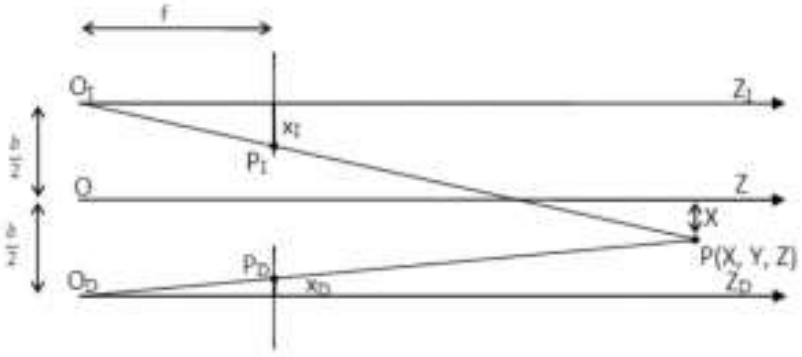


Figura 4.20: Calculo distancia focal.

Considerando igual la distancia focal *f* en las dos cámaras y conociendo la distancia entre cámaras *b.*

$$\begin{cases} O_i = \frac{\frac{b}{2}+X}{Z} = \frac{x_i}{f} \\ O_D = \frac{\frac{b}{2}-X}{Z} = \frac{x_D}{f} \end{cases} \quad \begin{cases} x_i = \frac{f}{Z}\left(\frac{b}{2}+X\right) \\ x_D = \frac{f}{Z}\left(\frac{b}{2}-X\right) \end{cases} \quad \boxed{d = x_i - x_d = \frac{jb}{Z}}$$

(1)

Como vemos, a partir del sistema anterior podemos obtener las coordenadas del punto *P*.

En resumen, Leap Motion funciona de la siguiente manera:

1. Obtiene las imágenes desde los sensores de las cámaras del dispositivo.
2. Aplica una corrección de la distorsión que producen los sensores.
3. Aplica un modelo para determinar la configuración de cada mano y ejecuta un algoritmo de visión estereoscópica entre cada par de imágenes para obtener la posición en el plano tridimensional.

ANALIZANDO LA API

En este apartado describiremos la SDK. Para ello vamos a ver sus partes fundamentales e intentaremos analizarlas ayudándonos de figuras. [1]

Nosotros vamos a analizar la API desde el punto de vista del lenguaje de programación C++, pero Leap Motion se puede programar en los siguientes lenguajes de programación y plataformas de desarrollo mostrados en la figura 4.21:

Figura 4.21: Lenguajes programación admitidos.

Desde la API se puede obtener todo tipo de información tridimensional referente a antebrazos, manos, herramientas, dedos e incluso a los “huesos” de los dedos que se tratan como objetos.

Visión general

El dispositivo utiliza el sistema de coordenadas cartesianas (Figura 4.22), en donde, desde la posición del usuario, el eje Y tiene valores positivos hacia arriba, el X hacia la derecha y el Z hacia el usuario.

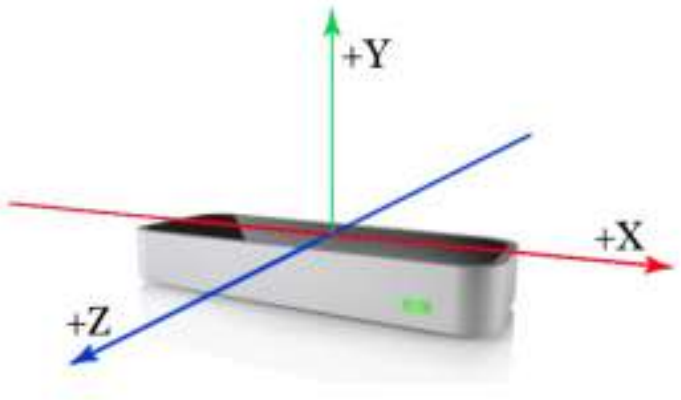


Figura 4.22: Sistemas de coordenadas.

La API mide magnitudes físicas con las siguientes unidades:

- **Distancia → mm**
- **Tiempo → µs**
- **Velocidad → mm/s**
- **Ángulos → radianes**

A continuación, iremos viendo los objetos principales que nos proporciona el dispositivo para reconocer, junto con el modelo anatómico que se usa para identificar cada uno, incluso analizaremos las clases principales que se describen en la API.

Componentes principales

El principal objeto es *Controller(),* se encarga de hacer de interfaz entre la aplicación a desarrollar y el dispositivo. Podemos observar los métodos que contiene:

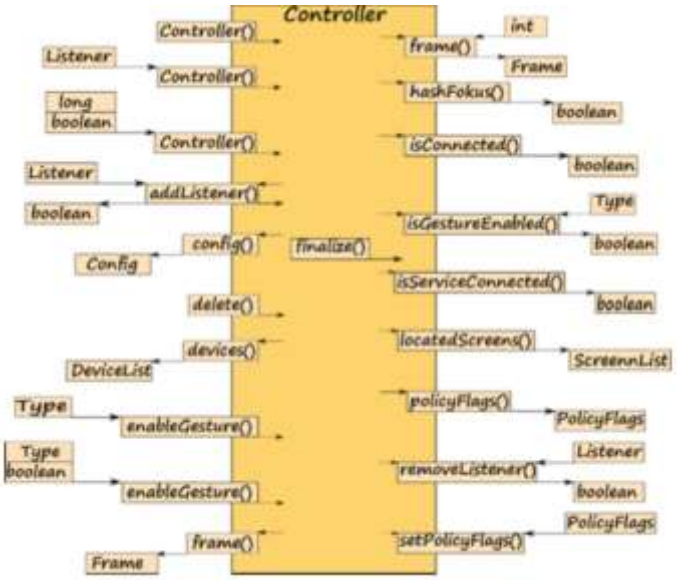


Figura 4.23: Objeto Controller.

El más interesante es el método *frame(),* a través del cual podemos acceder al objeto *Frame* que queramos, por defecto, accedemos al último que ha llegado. Se puede asociar a un objeto de tipo *Listener* al objeto *Controller* para recoger los eventos que ocurren en dicho *Controller*.

En la siguiente imagen podemos ver los eventos que se producen en el *Controller* y pueden ser obtenidos por *Listener:*

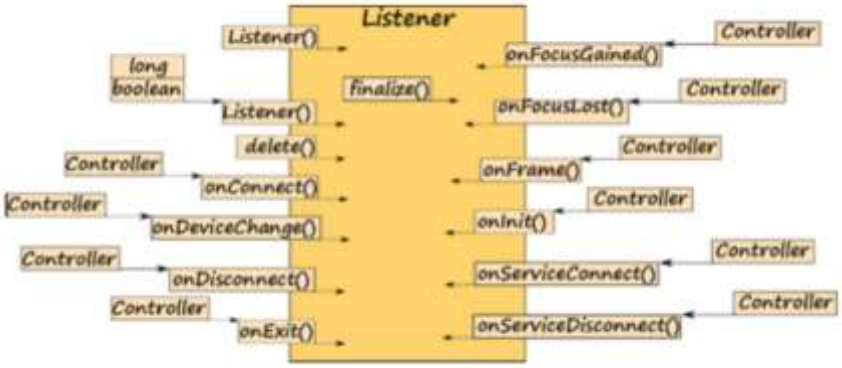


Figura 4.24: Objeto Listener

Hay diferentes eventos asociados al dispositivo hardware, es decir, a *Controller*. Otro método interesante es *onFrame(),* que se produce cada vez que el dispositivo captura una imagen. Este es el evento principal, dentro de este podemos asociar todos los objetos que más adelante veremos (manos, dedos, gestos etc).

Tenemos dos modos de acceder a un objeto *Frame*:

1. Desde el objeto *Controller*, a través del metodo *frame().*
2. Desde el objeto *Listener*, a través de los eventos que se van obteniendo.

La decisión de usar un modo u otro dependerá de si queremos analizar todos los frames o no que lleguen desde *Controller.* Si elegimos la opción de analizarlos todos, accederemos a la información del objeto *Frame* a través de *Listener.* Sin embargo si queremos analizar un objeto *Frame* cada cierto tiempo, se accede al objeto *Controller..(Ver detalles en figura 4.25).*

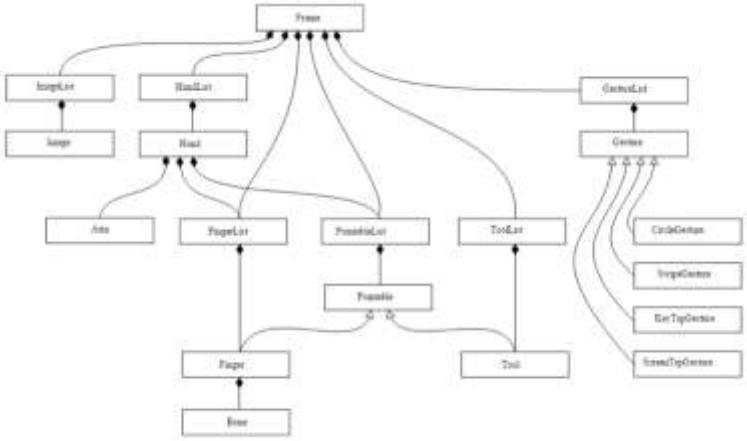


Figura 4.25: Diagrama Frame

Podemos observar que el objeto *Frame* es la raíz de todos los datos que analizamos en Leap Motion. *Frame* nos da acceso a todas las clases *List.*

Veamos ahora la lista de objetos *HandList.* Esta clase es una lista de objetos tipo *Hand,* que también contiene objetos de tipo: *Arm, Finger, Pointable* y *Tool*.

Dentro del objeto *Hand,* tenemos una lista de objetos *Finger* para cada *Hand* donde podemos acceder a información del objeto *Finger* que aparece en cada *Frame.* Incluso cada objeto de tipo *Finger* incluye objetos de tipo *Bone*. Ademas, los objetos *Finger y Bone* son también de tipo *Pointable.*

A continuación podemos ver un esquema de las partes que componen el objeto *Hand* con su representación anatómica:

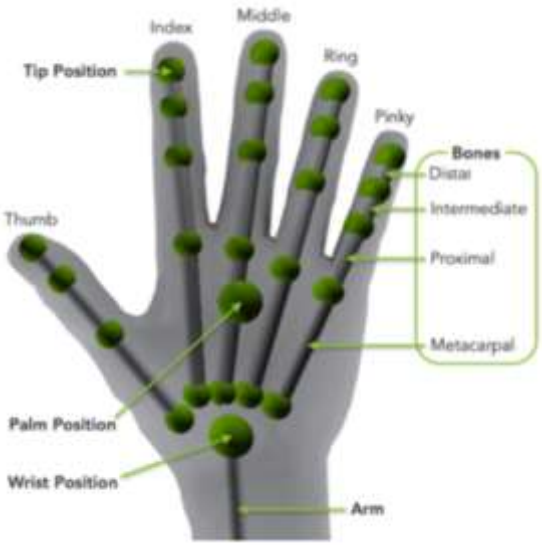


Figura 4.26: Representación anatómica.

La información principal que se puede extraer de cada objeto que depende de *Hand sería*:

Información principal incluida en el objeto *Hand.*

- Posición y velocidad de la palma.
- Dirección y vectores normales.
- Base ortonormal.

Información incluida en el objeto *Finger*

- Posición y velocidad de la punta.
- Vector de dirección.
- Base ortonormal.
- Anchura y longitud.

Información incluida en el objeto *Bone.*

- Posición de la articulación.
- Base ortonormal.
- Anchura y longitud.

Información incluida en el objeto *Arm.*

- Posiciones de muñeca y codo.
- Vector de dirección.
- Base ortonormal.
- Anchura y longitud.

Información incluida en el objeto *Pointable Tool.*

- Posición y velocidad de la punta.
- Vector de dirección.
- Base ortonormal.
- Anchura y longitud.

La siguiente clase a analizar (Figura 4.27) es *HandList*, que es la encargada de almacenar los objetos *Hand*:

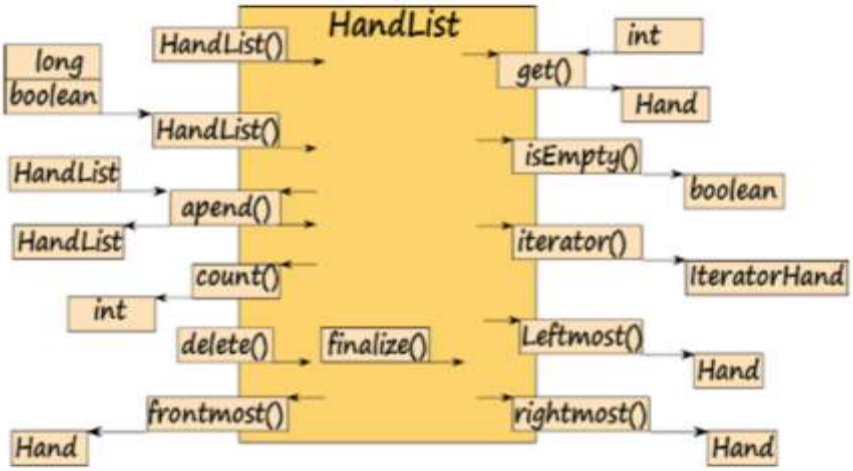


Figura 4.27: Clase HandList.

De esta clase se puede extraer un objeto de tipo *Hand* mediante el método *get().* También, se puede saber que objeto de tipo *Hand* está más cerca del dispositivo, la izquierda o la derecha.

Ahora veamos la figura 4.28, el objeto *Hand:*

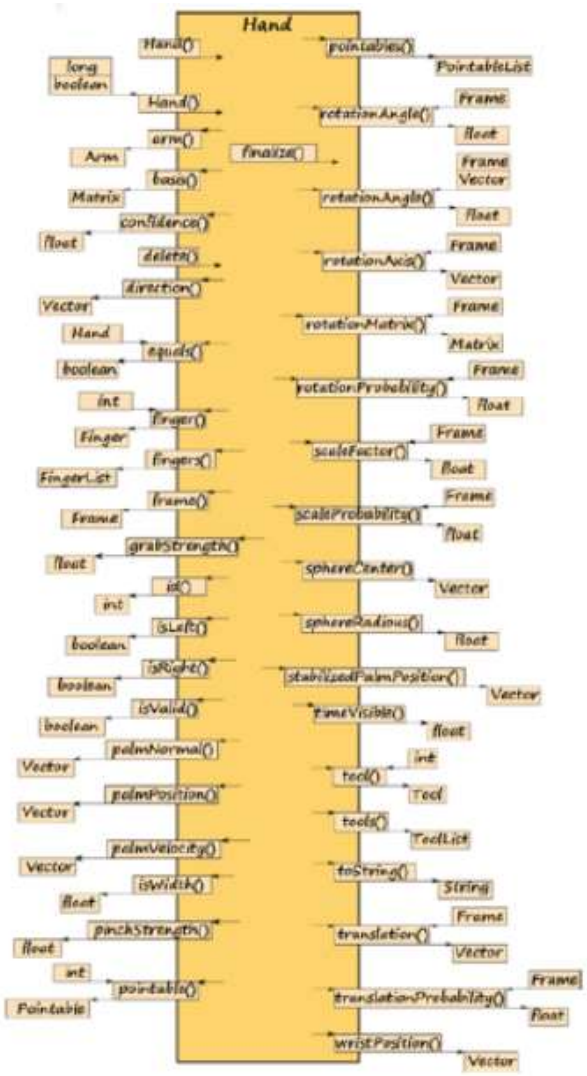


Figura 4.28: Objeto Hand

Se puede observar que se puede extraer bastante información referente al objeto *Hand.* Podemos acceder a los objetos *Finger y Arm*, es decir tipo de dedo y antebrazo de cada mano que detecte en la zona de detección.

Este objeto puede extraer información basándose en el modelo anatómico de la mano. (Ver figura 4.29).

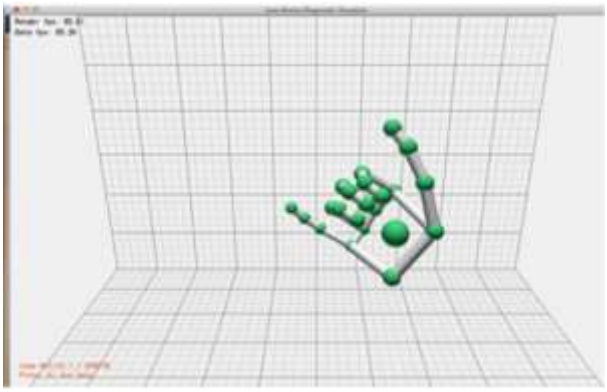


Figura 4.29: Simulación mano.

El modelo de la mano se basa en una estructura de puntos conectados a través de líneas que describen mejor la mano. Con este modelo aparece el objeto *Bones* que ha ayudado a que el dispositivo tenga mejor precisión. De forma que si no ve alguno de ellos puede determinar dónde está interpretando los demás que sí ve.
También podemos acceder a la información de cada dedo. Usaremos la clase *FingerList* que mostraremos a continuación:

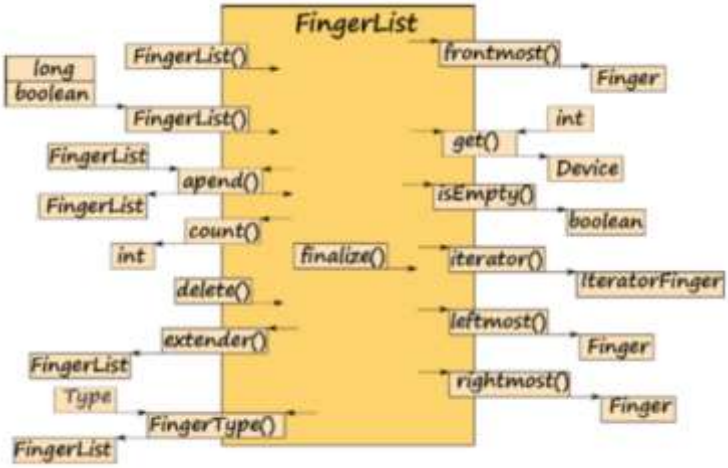


Figura 4.30: Clase FingerList.

Esta clase tiene mucha similitud a *HandList*, porque contiene objetos de tipo *Finger*. A parte podemos obtener información del objeto *Finger* más cercano al dispositivo, ya sea la derecha o la izquierda.

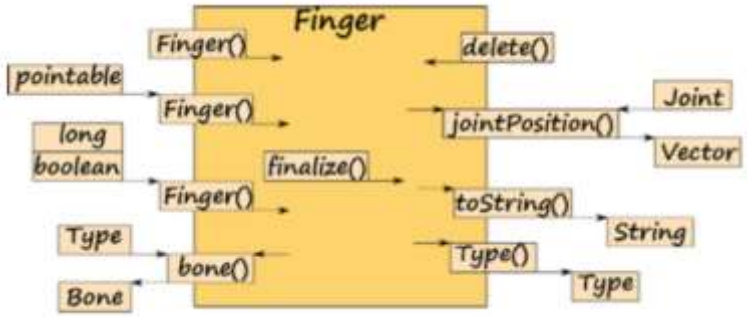


Figura 4.31: Objeto Finger

Este objeto, en el modelo, equivaldría a la siguiente figura:

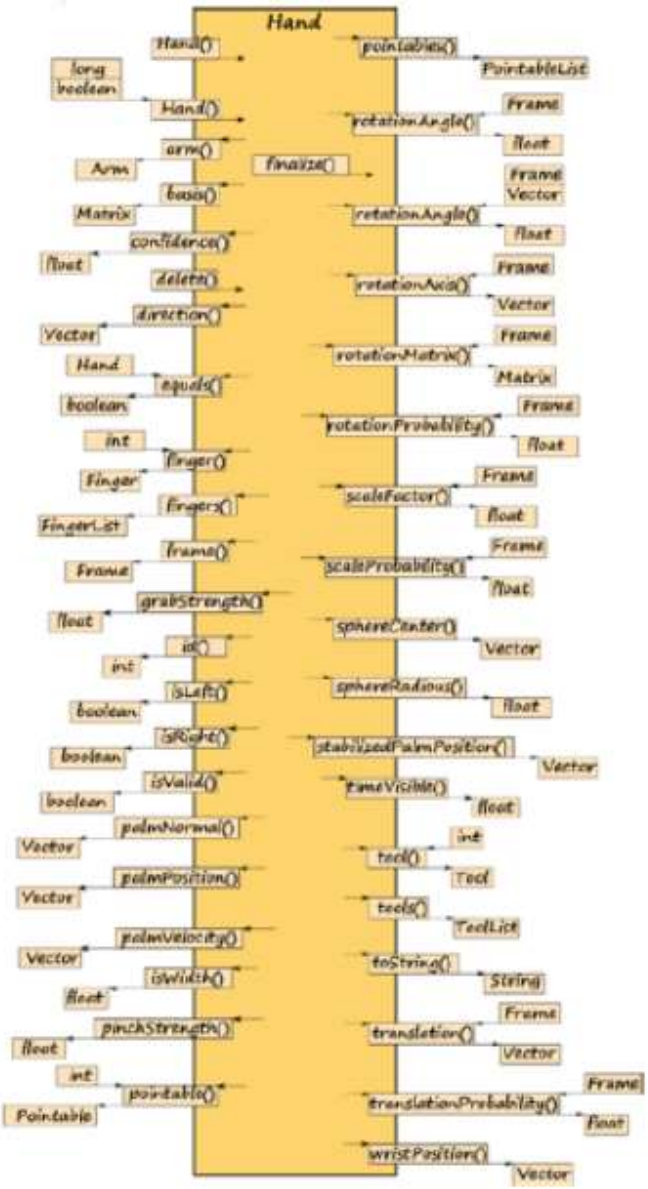


Figura 4.32: Funciones Hand.

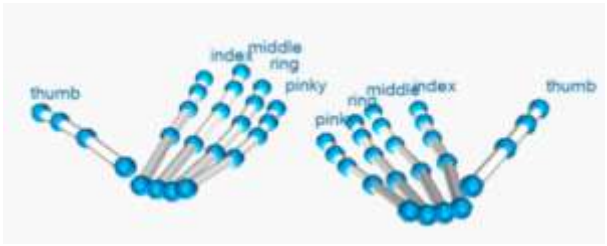


Figura 4.33: Huesos de la mano.

Tenemos incluso la funcionalidad de extraer de cada dedo, diferentes objetos *Bone*. Estos objetos representan los puntos que definen al dedo y estarían representados de la siguiente manera:

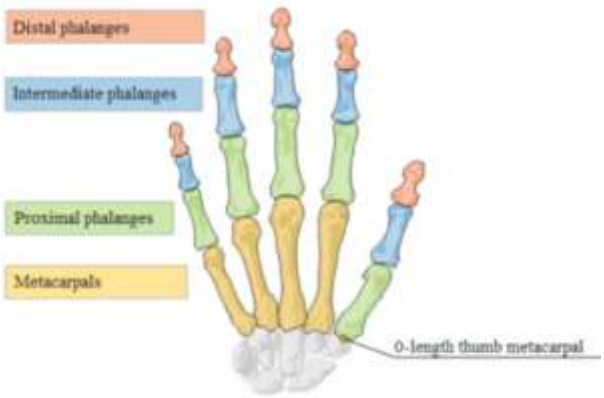


Figura 4.34: Huesos de la mano (real).

Como vemos, cada objeto *Bone* puede ser de un tipo que será el que determine su identificador (Ver detalles en figura 4.35). Podemos ver mejor como es interpretado esto por la API:

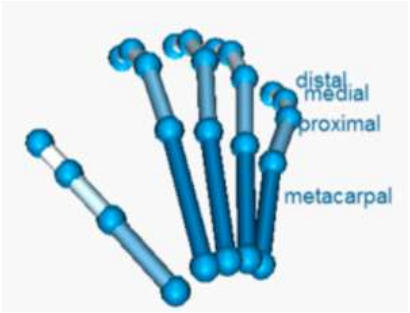


Figura 4.35: Partes de cada hueso.

Por último, en la figura 4.36 vemos el objeto Bone:

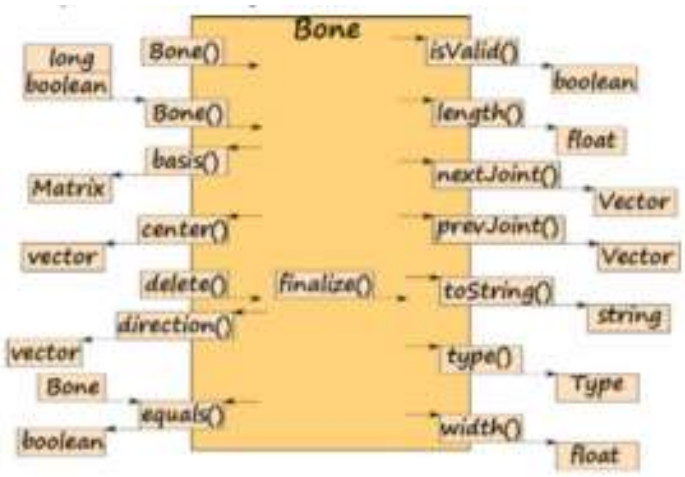


Figura 4.36: Objeto Bone.

Esta clase incluye bastante información de cada hueso. Gracias a este modelo matemático, es posible determinar mejor la posición de cada objeto y direcciones. Esto llevó a la aparición de un nuevo objeto, *Arm:*

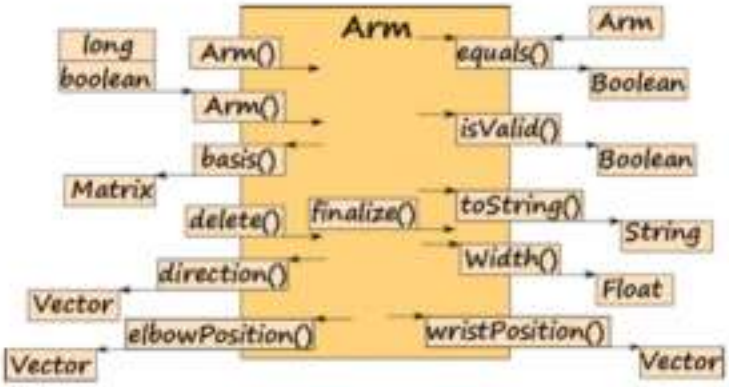


Figura 4.37: Objeto Arm.

Además, la API también nos provee de una serie de objetos que ayudan a poder extraer los datos que necesitemos, como por ejemplo el objeto *Vector*.

Conclusión

Como vemos, el dispositivo Leap Motion cuenta con una API muy extensa y optimizada para la extracción de datos característicos de las partes que forman una mano. Por lo tanto, es un dispositivo óptimo para nuestro proyecto.

4.2 OGRE3D

En su blog Juan Antonio Bolós Miguel [7], nos introduce a Ogre, es el acrónimo del inglés: Object-Oriented Graphics Rendering Engine; se trata de un motor de renderizado de gráficos en tres dimensiones y en tiempo real. Está programado en C++ y es software libre con licencia MIT.

Fue creado por Steve Streeting (también conocido como Sinbad) en 2001 con el propósito de crear un componente de renderizado en tiempo real sin hacer asunciones a nivel de aplicación. El objetivo era crear un componente genérico que pudiese ser ampliado a través de plugins. Desde un principio el proyecto se diseñó teniendo en cuenta la mantenibilidad y la facilidad de ampliación.

Ogre3D no fue concebido como un motor de juegos. Se pretendía cubrir el mayor espectro posible, de manera que no sólo sirviese a campos como el de los videojuegos, sino también a los de simulación, realidad aumentada, realidad virtual,...y en general, cualquier campo que requiriese del uso de herramientas de renderizado en tiempo real.
Aquí os muestro unas par de imágenes (Figura 4.38 y 4.39) de lo que podemos conseguir con Ogre:

Figura 4.38: Ejemplo OGRE 1.

Figura 4.39: Ejemplo OGRE 2.

Además, el hecho de que se distribuya bajo una licencia de código libre contribuye muchísimo más a su éxito. Esto es así debido a que la comunidad está muy involucrada con el proyecto, cosa que podemos observar en el foro oficial de éste, donde se resuelven dudas de desarrollo, se discute el roadmap, etc. En cuanto a las políticas de contribuciones, los usuarios de la comunidad pueden colaborar bien realizando pull-request al repositorio oficial con sus parches o bien reportando bugs al Jira del proyecto

Como nos indica Isaac Lacoba [8], Ogre3D no es un motor de juego. Esto implica que será el desarrollador quien tenga que encargarse de aspectos como la gestión de eventos de entrada (teclado, ratón,...), físicas, networking, interfaces, etc. En el caso del desarrollo de interfaces existen maneras de crearlas con Ogre a través del uso de overlays; sin embargo, esta aproximación no es lo suficientemente flexible como para crear interfaces avanzadas. Las características principales de Ogre son:

- **Mutiplataforma**: permite el desarrollo para sistemas Windows, GNU/Linux y Mac OS X.
- **Diseño a alto nivel**: Ogre3D encapsula llamadas a las librerías gráficas DirectX y OpenGL. Además, hace uso de patrones de diseño: *observer* para informar de eventos y cambios de estado, *singleton* para evitar que exista más de una instancia de cualquier manager, *visitor* para realizar operaciones sobre un objeto y evitar modificarlo (por ejemplo, en los nodos del grafo de escena), *façade* para unificar el acceso a operaciones, *factory* para creación de objetos concretos de interfaces abstractas, etc.
- **Grafo de escena**: una de las características más importantes del grafo de escena de Ogre es que desacopla el propio grafo del contenido de la escena, definiendo una arquitectura de plugins. A diferencia de otros motores gráficos, como Irrlicht3D, Blitz3D o Unreal, Ogre no se basa en la herencia como principio de diseño del grafo, sino en la composición. Esto permite expandir el diseño para soportar otros tipos de datos, como audio o elementos de simulación física. En la siguiente figura podemos ver el esquema general del grafo de escena de Ogre.

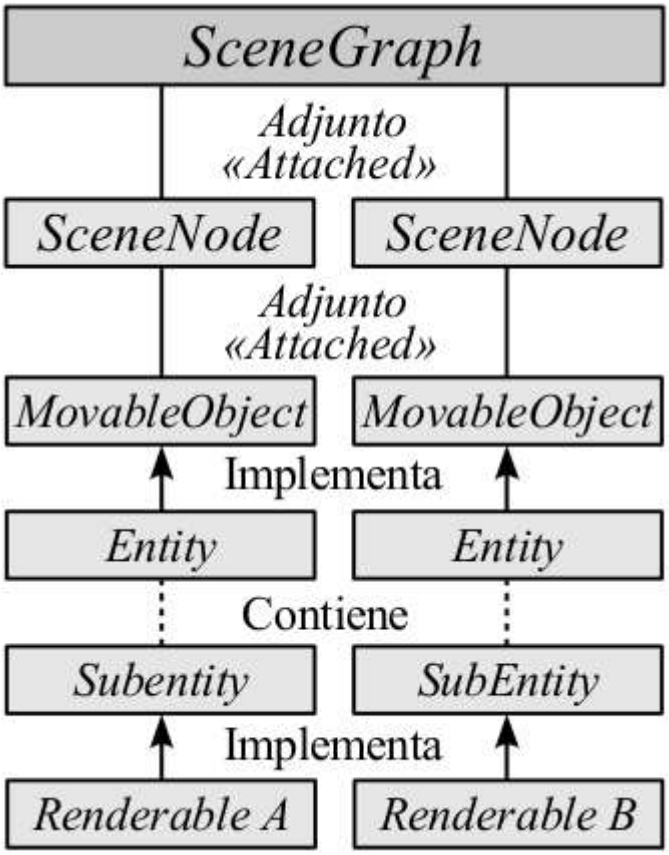


Figura 4.40: Grafo de una escena.

- **Aceleración Hardware**: OGRE permite definir el comportamiento de la parte programable de la GPU mediante la definición de Shaders, estando al mismo nivel de otros motores como Unreal o CryEngine.

- **Materiales**: se definen mediante un sistema de scripts y permiten asignar o cambiar los materiales de los elementos de la escena sin modificar el código fuente.

- **Animación**: tres tipos (skeletal, morph y pose). La animación y la geometría asociada a los modelos se almacena en un único formato binario optimizado. El proceso más empleado se basa en la exportación desde la aplicación de modelado y animación 3D a un formato XML (Ogre XML) para convertirlo posteriormente al formato binario optimizado mediante la herramienta de línea de órdenes OgreXMLConverter.

- **Composición y Postprocesado**.
- **Plugins**.
- **Gestión de Recursos**: Ogre ofrece una serie de gestores de recursos, organizados jerárquicamente por grupos.
- **Características específicas avanzadas**: El motor soporta gran cantidad de características de visualización avanzadas, tales como sombras dinámicas (basadas en diversas técnicas de cálculo), sistemas de partículas, animación basada en esqueletos y de vértices, y un largo etcétera. OGRE soporta además el uso de otras bibliotecas auxiliares mediante plugins y conectores. Entre los más utilizados cabe destacar las bibliotecas de simulación física ODE, el soporte del metaformato Collada, o la reproducción de streaming de vídeo con Theora.

4.3 DTW

En este trabajo [10] se presenta un nuevo algoritmo que ha sido específicamente diseñado para el reconocimiento temporal multivariante de gestos. El algoritmo se basa en Dynamic Time Warping y se ha extendido a clasificar cualquier señal N dimensional, calcular automáticamente una clasificación umbral para rechazar cualquier dato que no es un gesto válido y ser rápidamente entrenado con un bajo número de ejemplos de entrenamiento.

4.3.1 Introducción

Dynamic Time Warping es un algoritmo que puede calcular la similitud entre dos series de tiempo, incluso si las longitudes de la serie temporal no coinciden. Uno de los principales problemas usando una medida de distancia (por ejemplo, la distancia euclidiana) a medir la similitud entre dos series de tiempo es que los resultados a veces pueden ser muy poco intuitivos. Si, por ejemplo, dos series de tiempo son idénticas, pero ligeramente desfasados entre sí, a continuación, una medida de distancia como la distancia euclidiana dará una medida de similitud muy pobre.

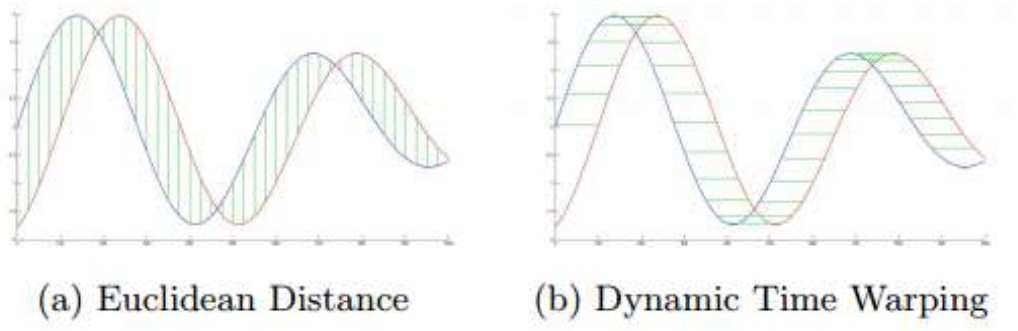


Figura 4.41: Calculo de las posibles distancias.

4.3.2 DTW unidimensional:

El algoritmo de base para DTW es el siguiente. Dado dos ejemplos, unidimensional, de series de tiempo , x = {x1 , x2, ... , $x_{|x|}$ } e y = { y1, y2 , ..., $y_{|y|}$} . Con longitudes respectivas | x | e | y |, construir un camino de deformación w = {w1, w2,..., $w_{|w|}$} asi que |w| , la longitud de W es :

$$Max\{|x|,|y|\} \leq |W| < |X| + |Y| \quad (2)$$

donde el valor k-ésimo de w está dada por:

$$w_k = (x_i , y_j) \quad (3)$$

Una serie de limitaciones se para construir el camino de deformación, que son los siguientes:

- La trayectoria de deformación debe comenzar en: $w_1 = (1, 1)$
- La trayectoria de deformación debe terminar en: $w_{|w|} = (|x|, | y |)$

- La trayectoria de deformación debe ser continua , es decir, si w_k =(i, j), entonces w_{k+1} debe ser igual a o bien (i , j) , (i + 1 , j) , (i, j + 1) o (i+1, j + 1).

- La trayectoria de deformación debe exhibir un comportamiento monótono, es decir, el camino de deformación no puede moverse hacia atrás.

Hay exponencialmente muchos caminos de deformación que satisfacen las condiciones anteriores. Sin embargo, sólo estamos interesados en encontrar el camino que minimiza el costo total, dado por:

$$\min \frac{1}{|w|} \sum_{k=1}^{|w|} DIST\ (w_{k_i},\ w_{k_j}) \qquad (4)$$

Donde DIST ($w_{k,i}$, $w_{k,j}$) es la función de distancia (típicamente Euclidiana) entre el punto i en las series de tiempo x y el punto j en series de tiempo y , dado por w_k . La ruta de mínima deformación se puede encontrar mediante el uso de programación dinámica para rellenar una matriz bidimensional de coste C (| x | | y |). Cada celda de la matriz de costos representa la deformación mínima acumulada hasta ahora en la ruta entre la serie de tiempo x e y hasta la posición de esa celda. El valor de la celda en C ($_{i, j}$) por lo tanto, está dada por:

$$C_{(i,j)} = DIST_{(i,j)} + \min\{ C_{(i-1,j)}, \quad C_{(i,j-1)}, C_{(i-1,j-1)}\} \qquad (5)$$

Es la distancia entre el punto i de la serie de tiempo x y el punto j en la serie temporal y, más la mínima distancia calculada de las tres celdas anteriores a ese vecino de la celda i,j (la celda por encima de ella , la célula a su izquierda y la célula en su diagonal) . Cuando la matriz de costos se ha llenado, la ruta de deformación mínima posible se puede calcular fácilmente navegando a través de la matriz de coste en orden inverso, comenzando en C ($_{| x |, | y |}$), hasta que se ha alcanzado la celda C ($_{1,1}$), como se ilustra en la Figura 2. En cada paso, la celda a la izquierda, por encima y en diagonal de la celda actual se buscan para encontrar el valor mínimo. La célula con el valor mínimo es entonces trasladada y las anteriores tres celdas se repite hasta que se ha alcanzado C ($_{1,1}$). La ruta de la deformación a continuación da la distancia mínima total entre x e y:

$$\text{DTW (x,y)} = \frac{1}{|w|} \sum_{k=1}^{|w|} DIST\ (w_{k_i},\ w_{k_j}) \qquad (6)$$

Aquí, $\frac{1}{|w|}$ se utiliza como un factor de normalización para permitir la comparación de series de tiempo de distintas longitudes.

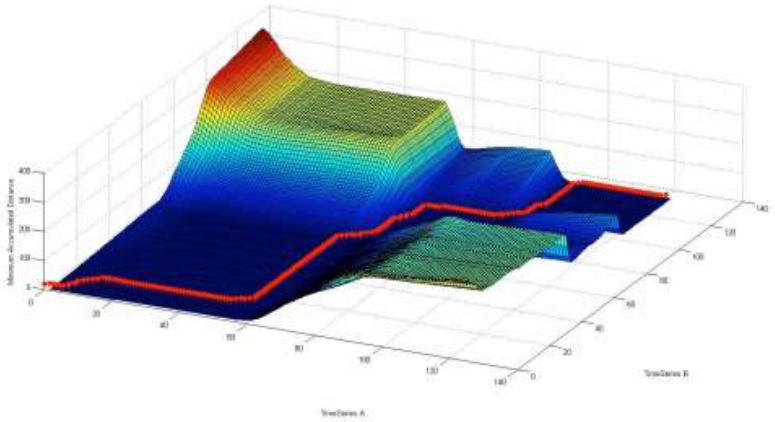


Figura 4.42: Construcción del camino más corto para la identificación del gesto.

4.3.4 Reducción numérica:

DTW es una herramienta útil para el cálculo de la distancia entre dos series de tiempo. Es, sin embargo, un algoritmo computacional costoso para usar para el reconocimiento en tiempo real, ya que cada valor en la matriz de costos tiene que ser completada. Es evidente que esto no se puede utilizar para tiempo real con fines de reconocimiento, sobre todo si las series de tiempo se están emparejado contra una gran base de datos de los gestos. Para acelerar tanto la formación de las plantillas y de la clasificación en tiempo real de gestos desconocidos, N- dimensionales, hemos probado varios métodos numéricos de reducción. Quizás uno de los métodos más rudimentarios para la reducción de numerosidad es reducir la resolución de la serie temporal en un factor de n. Para evitar el aliasing, los datos se filtran utilizando un filtro FIR de paso bajo con una ventana rectangular y un filtro orden de n.

4.3.5 Restringir el camino de deformación:

Otro método comúnmente adoptado para mejorar la eficiencia de DTW es restringir la ruta de deformación de manera que la trayectoria máxima deformación permitida no puede desviarse demasiado lejos de la diagonal. Controlando el tamaño de esta ventana afectará en gran medida la velocidad de la computación DTW. Si la ventana de deformación es pequeña, una gran proporción del coste la matriz no tiene

que ser buscado o incluso construido. El tamaño de la ventana de deformación puede ser controlada mediante la variación de el parámetro r, dado como el porcentaje de la longitud de las series temporales de la plantilla. La ventana de la deformación establece como la distancia, r, de la diagonal directamente encima y a la derecha de la diagonal. Este tipo de restricción global que se conoce como la banda Sakoe - Chiba. Itakura también se ha propuesto otra restricción global basada en un paralelogramo.

4.3.6 ND-DTW

Anteriormente hemos descrito la implementación estándar de DTW para dos series de tiempos unidimensionales. Es común, sin embargo, en los campos de cálculo, tales como el reconocimiento de gestos para tener series de tiempo con múltiples dimensiones, como datos capturados por un acelerómetro de 3 ejes. En este caso se requiere una implementación de DTW que pueda calcular la distancia entre dos series de tiempo N- dimensionales. Vamos a utilizar el enfoque común utilizado para calcular la distancia entre dos series de tiempo N- dimensionales. Esto toma la suma de los errores de distancia entre cada dimensión de una plantilla de N- dimensional y la nueva N - dimensional de series de tiempo. La distancia total a través de todos las N dimensiones se utiliza entonces para construir la matriz de deformación C. Se utilizará la distancia euclidiana como una distancia medida a través de las dimensiones de N de la plantilla y la nueva series de tiempo.

$$DIST\ (i,j) = \sqrt{\sum_{n=1}^{N} (i_n - j_n)^2} \tag{7}$$

La siguiente sección describe nuestra N - dimensional Dynamic Time Warping (ND-DTW). En la etapa de entrenamiento, se calcula para cada una de las plantillas N dimensionanles, (φ_g) y el umbral (τ_g) para cada uno de los gestos G. En la etapa de predicción en tiempo real una nueva serie temporal de N-dimensiones se clasifica con el de la plantilla que da la mínima distancia normalizada entre la plantilla N –

dimensional y la serie temporal de N dimensión desconocida. Ahora se discutirá cada elemento del algoritmo en detalle.

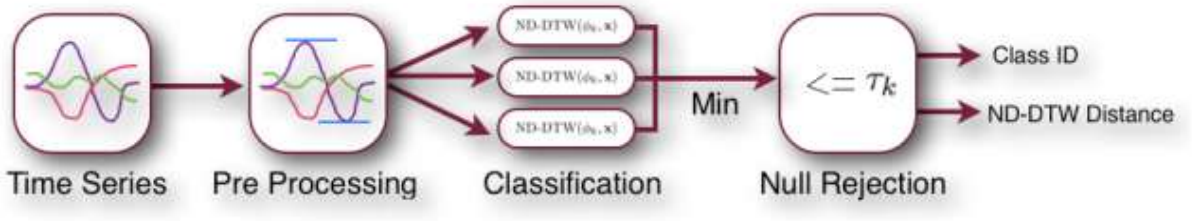


Figura 4.43: Esquema de procesamiento de los datos.

4.3.7 Entrenando ND-DTW

Con el fin de que, ND- DTW, sea utilizado como un algoritmo de reconocimiento en tiempo real, primero se debe crear una plantilla para cada gesto que necesita ser clasificado. Una plantilla se puede calcular mediante el registro de ejemplos de entrenamiento M_g para cada uno de los gestos G que se requieren para ser reconocido. Después de que los datos de entrenamiento se hayan grabado, cada una de las plantillas de G se puede encontrar mediante el cálculo de la distancia entre cada uno de los M_g ejemplos de entrenamiento para el gesto g_{th} y buscar el ejemplo de entrenamiento que proporciona la distancia mínima normalizada de deformación cuando se compara con la otra formación de $M_g - 1$ de esa clase. Por consiguiente, la plantilla g_{th} (φ_g) está dada por:

$$\phi_g = \arg\min \frac{1}{M_g - 1} \sum_{J=1}^{M_g} 1\{ND - DTW(X_i, X_j)\} \qquad 1 \leq i \leq M_g \qquad (8)$$

Donde la 1 {·} que rodea la función ND- DTW es el soporte del indicador , dando 1 cuando $i \neq j$ o 0 en caso contrario y X_i e X_j son los ejemplos de entrenamiento n-dimensional i_{th} y j_{th} por el gesto. La función de ND- DTW en (9) es simplemente la extensión del algoritmo estándar de DTW a N- dimensiones:

$$\text{ND-DTW (x,y)} = \frac{1}{|w|} \sum_{k=1}^{|w|} DIST\ (w_{k_i},\ w_{k_j}) \tag{9}$$

$$DIST\ (i,j) = \sqrt{\sum_{n=1}^{N} (i_n - j_n)^2} \tag{10}$$

4.3.8 Entrenamiento de subprocesos:

Una gran ventaja de utilizar el algoritmo DTW es que cada plantilla (es decir, cada gesto) se puede calcular de manera independiente de las otras plantillas. Esto es debido a que las máquinas nuevas cuentan con múltiples procesadores como un enfoque de entrenamiento de múltiples subprocesos que puede ser adoptada en la rutina de entrenamiento de cada plantilla. Este enfoque de entrenamiento acelera en gran medida el tiempo de formación de un sistema de clasificación, una plantilla DTW no tiene que esperar a que la plantilla anterior sea entrenada antes de que pueda comenzar su propia rutina de entrenamiento.

El algoritmo DTW también tiene otra ventaja es que, si se añade un nuevo gesto a un modelo de entrenamiento existente o uno existente se elimina, todo el modelo no necesita de nuevo entrenamiento. En lugar de ello, una nueva plantilla y el umbral es necesario para entrenar nuevo gesto, por tanto, reduciendo en gran medida el tiempo de entrenamiento. Si un gesto existente se borra del modelo entonces no se requiere re-formación porque el sistema de clasificación DTW simplemente elimina esta plantilla y el valor de umbral de su "base de datos". Esto no es en el caso de otros algoritmos de aprendizaje automático, como una Red Neuronal Artificial, ya que todo el sistema necesitaría ser re-entrenado desde cero cada vez que se añade un nuevo gesto o eliminado.

CLASIFICACION USANDO ND-DTW

Después de que el algoritmo DTW - ND ha sido entrenado, una desconocida serie temporal N-dimensional X, se puede clasificar por el cálculo de la distancia total de deformación normalizada entre X y cada una de las plantillas de G en el modelo **C.**

El índice de clasificación que representa el gesto es la plantilla correspondiente que dio la mínima distancia total de deformación normalizada:

$$c = \arg\min \text{ND-DTW}(\phi_g, X) \qquad 1 \leq g \leq G \tag{11}$$

4.3.9 Determinar el umbral de clasificación:

Utilizando la ecuación (11) **X**, una desconocida serie temporal N-dimensional, se puede clasificar mediante el cálculo de la distancia entre ella y todas las plantillas en el modelo. La serie temporal desconocida **X** entonces se puede clasificar comparando con la plantilla que resulta de la distancia total de deformación normalizada más baja. Este método, sin embargo, da falsos positivos si X está compuesto por varios de los gestos en el modelo. Este problema de clasificación puede ser mitigado mediante la determinación de un umbral de clasificación para cada gesto de la plantilla durante la fase de entrenamiento. En la fase predicción, un gesto únicamente se califica en con la plantilla que da como resultado la distancia de deformación total mínima normalizada, si esta distancia es menor que o igual a la del umbral de la clasificación del gesto. Si la distancia está por encima el umbral de clasificación, entonces el algoritmo clasificará el gesto contra una clase nula, lo que indica que no pudo encontrar coincidencias:

$$\hat{c} = \begin{Bmatrix} c \;\; if(d \leq \tau_g) \\ 0 \; en \; cualquier \; otro \; caso \end{Bmatrix} \tag{12}$$

Donde **c** es dada por la ecuación (12), d la distancia total de deformación normalizada de entre φ_g y X, e τ_g es la umbral de clasificación para la plantilla g_{th}. El umbral de clasificación para cada plantilla se puede ajustar como la distancia deformación total normalizada promedio entre φ_g y los otros ejemplos de entrenamiento M_g-1 para ese gesto más la desviación gamma estándar:

$$\tau_g = \mu_g + (\sigma_g \gamma) \tag{13}$$

donde

$$\mu_g = \frac{1}{M_g - 1} \sum_{J=1}^{M_g} 1\{ND - DTW(\phi_g, X_i)\} \tag{14}$$

$$\sigma_g = \sqrt{\frac{1}{M_g - 2} \sum_{J=1}^{M_g} 1\{(ND - DTW(\phi_g, X_i) - \mu_g)^2\}} \tag{15}$$

donde la 1 {·} que rodea la función ND- DTW es el soporte de indicador , dando 1 cuando i ≠ del índice del ejemplo de entrenamiento que dio la distancia total mínima normalizada cuando se compara con la otra M_g-1 ejemplos en esa clase (es decir, la plantilla) o 0 en caso contrario y X_i es el ejemplo de entrenamiento para la clase i-ésima g_{th} . γ se puede ajustar inicialmente a un número de desviaciones estándar (Por ejemplo, 2) durante la fase de entrenamiento y luego se ajuste por el usuario en la fase de predicación en tiempo real hasta que un adecuado nivel clasificación se ha logrado .

Este es crítico para el cálculo del umbral de la clasificación para cada uno de los gestos g para realizar cualquier procesamiento previo tales como la ampliación o reducción de la resolución en el mismo orden en que lo realizaría durante la fase de clasificación en tiempo real. Si esto no se completa en el mismo orden entonces el umbral óptimo de clasificación no será encontrado. Ahora discutiremos las diversas opciones de pre procesamiento que se pueden utilizar para ND-DTW.

4.3.10. Pre-procesamiento ND-DTW

El pre-procesamiento es necesario para ND- DTW si o bien cualquier de los datos de N- dimensiones se originan de una fuente de diferente rango o si la invariancia a la variabilidad espacial y variabilidad de magnitud de la señal es diferente a la que se

desea. Ahora discutimos estos puntos y daremos soluciones apropiadas de pre-procesamiento para cada uno.

- *VARIACION DE RANGOS DE ENTRADA*

  Es importante que cada uno de los datos de X proceda de un origen común. Si este no es el caso, entonces una o más de las dimensiones pueden fuertemente ponderar los resultados de la DTW. Si cada uno de los datos N-dimensionales no provienen de una fuente común entonces, cada canal será escalado usando el mín-máx de normalización antes de tanto la formación de las plantillas y de la predicción en tiempo real.

- *INVARIANCIA DE AMPLITUD Y VARIABILIDAD ESPACIAL*

  Varianza espacial y variabilidad en la amplitud de la señal puede ser mitigado normalizando tanto las series de tiempo de entrada como también las plantillas de reconocimiento. La normalización, dará la entrada y la plantilla de series temporales de media cero y varianza unitaria, por lo tanto, la eliminación de cualquier efecto de variabilidad espacial o variabilidad en la amplitud de la señal.

5. ESTADO DEL ARTE

5.1 Trabajos relacionados

Basados en los trabajos [12] [13] introduciremos **tres estudios** sobre el reconocimiento de gestos:

El primero del que vamos a hablar, se trata del uso de guantes de colores para el reconocimiento de lenguaje de signos.
Se trata de guantes teñidos con 6 colores diferentes (Ver figura 5.1). Las vectores de características se obtienen utilizando el análisis de componentes principales de las imágenes capturadas.

Figura 5.1: Guantes usados para el reconocimiento óptico.

La detección de manos y dedos se realiza sobre un fondo con un color homogéneo. Como podemos ver en la imagen el negro. En este método no importa la lejanía de la cámara con las manos. Para analizar el rendimiento de este estudio se usaron redes neuronales para el reconocimiento. La red neuronal es feed-forward, que se caracteriza porque la conexión entre sus neuronas no forma un ciclo, es decir, la información solo va en una dirección.

El número de neuronas en las capas intermedia y de salida son 42 y 26, respectivamente. El resultado de este experimento, con 30 muestras de cada gesto, fue de un 93 %.

El segundo estudio, descrito en [ ], se propone un enfoque sólido usando una combinación nueva de características. Las características son el color de las imágenes, la profundidad de las imágenes y la forma de la mano.

Como podemos observar en la figura 5.2, la obtención de profundidad, color y dato del esqueleto obtenidos en Kinect, se divide en una imagen en color en una mano y una imagen de profundidad de la mano. Los vectores de características de color y profundidad se caracterizan por el patrón local binario (LBP) calculado a partir de la imagen de la profundidad de la mano de imagen y el color mano. También podemos extraer la forma de la mano.

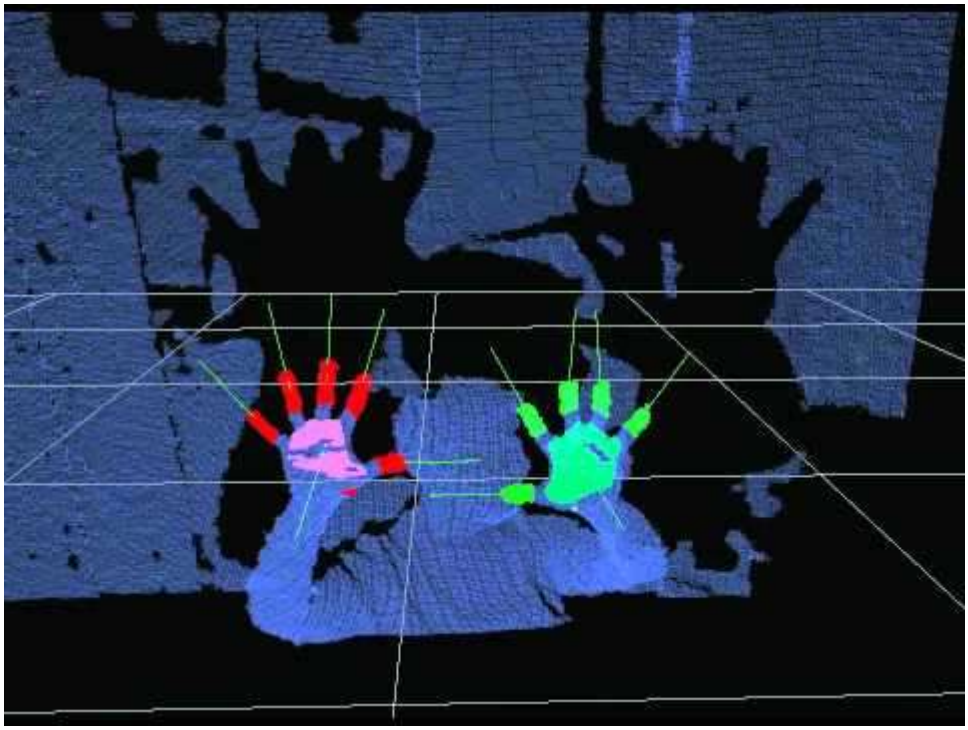


Figura 5.2: Visión del dispositivo Kinect.

La combinación de estos tres vectores de características se utiliza para el reconocimiento mediante comparación de plantillas y máquinas de soporte vectorial (SVM).
Dentro de este estudio se realizaron dos estudios. El primero solo tenía en cuenta por separado la profundidad, el color o la combinación de ambas. El segundo se realizó combinando a parte de las características mencionadas se incluyó la forma de la mano. Dieron un resultado de un 92% y 95% respectivamente.

Por último el tercer estudio, [ ] se basa en la representación de gestos basada en modelos 3D, y que se fundamenta en la detección de los movimientos de la mano mediante su seguimiento. Incluyen enfoques basados en modelos ocultos de Markov (HMM) , máquinas de estados finitos (FSM), tiempo de retardo de Redes Neuronales (TDNN).

Estos métodos utilizan una representación visual de los datos tales como las características locales espacio-temporales, trayectorias de movimiento y otros.

El primer método basado en modelos es **Modelos Ocultos de Markov**. HMM se considera como forma específica de red bayesiana dinámica y es doblemente proceso estocástico con un proceso estocástico subyacente que no es observable (está oculto), pero sólo se puede observar a través de otro conjunto de procesos estocásticos que producen la secuencia de símbolos observados.

Para las pruebas, se definió 9 gestos que representan los números del 1 al 9. Lograron una tasa de reconocimiento del 99,78 %.

FSM consiste en un número finito de estados, que uno de ellos está marcado como el estado actual. La transición entre los estados se realiza en el caso de que se produzca el evento correspondiente. FSM también se define por un conjunto estados y recogida de las condiciones de transición.

El seguimiento de los dedos se ha aplicado para determinar las rutas de movimiento y para encontrar el inicio y parada de las posiciones de los gestos.

Otro método es el **Tiempo de Retardo de Redes Neuronales** (TDNN) , que es un conjunto de redes neuronales con una arquitectura especial. La característica principal de este enfoque es que el reconocimiento es independiente del desplazamiento de tiempo de la muestra. TDNN opera en los datos secuenciales.

Se proporcionó la extracción y clasificación de gestos dinámicos bidimensionales. El uso de las trayectorias de movimiento, la segmentación de múltiples escalas y las transformaciones afines autores alcanzaron 96,21 % para el grupo de pruebas de los gestos de ASL.

5.2 Prototipos analizados

Este apartado consiste en una búsqueda y un análisis de los prototipos, librerías y programas que usan Leap Motion y que puedan cumplir con alguno de nuestros requisitos.

Estos programas podrán servirnos de referencia para la implementación de nuestro prototipo. En ellos podemos ver como se guardan los datos de los movimientos, y en algunos casos como se podrían comparar 2 gestos para saber si son similares o no.
También intentaremos buscar alguna implementación sobre Ogre3D para ayudarnos a combinar ambas tecnologías.

5.2.1 JestPlay:

JestPlay [14], programada en JavaScript, esta aplicación nos permite grabar nuestros gestos y movimientos guardando los datos que devuelve Leap Motion a nuestra máquina mediante BVH (Siglas) o JSON (siglas).
El formato de los datos en JSON pero con información adicional, como el número de dedos, la distancia entre cada dedo o los ángulos de cada miembro.

Al crear un movimiento, se obtiene un fichero con los datos, y estos se pueden introducir en el programa para que reproduzca el gesto visualmente. Por lo que al grabar la mano, la podemos reproducir en 3D (Ver figura 5.3).

Esta herramienta está orientada a priori solo para las aplicaciones web, aunque se podrían exportar y usar en diferentes plataformas.

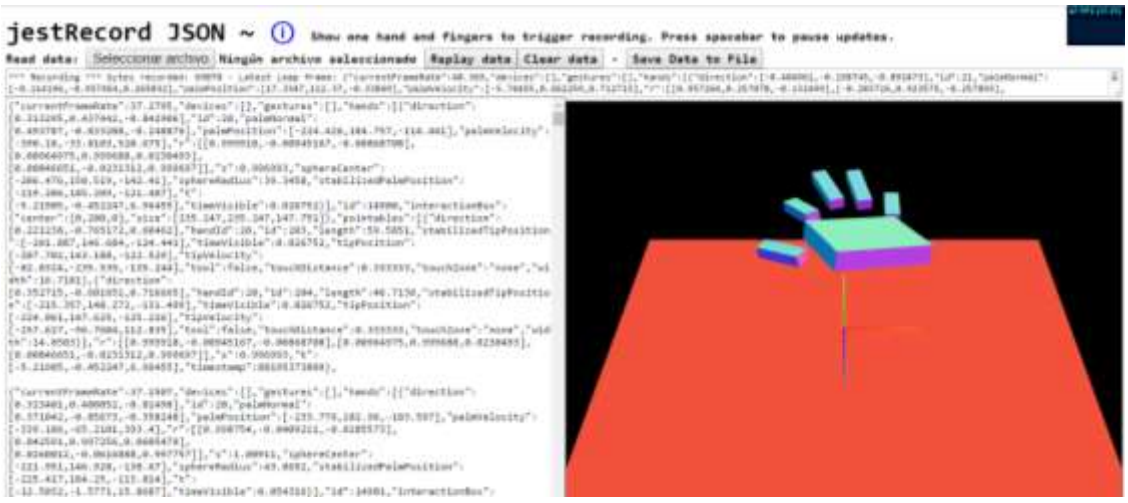


Figura 5.3: Interfaz JestPlay.

Con esto lo que se podría aprovechar serían los datos que se obtienen al grabar un movimiento, por lo que tendríamos que crear un comparador para evaluar los gestos que se realizan con los que se tienen grabados y sacar un porcentaje de acierto para saber si son similares.

5.2.2 HandVisualiser:

HandVisualiser [15] es un excelente proyecto que une la tecnología Leap Motion y Ogre3D. Está programado en C++ y es una herramienta sencilla para visualizar los datos de la mano que recoge el Leap Motion.
Crea un esqueleto de la mano adquiriendo información de los huesos de los dedos y la posición global de la mano. Además también recoge la información sobre la inclinación y posición.

Podemos visualizarlo en la siguiente imagen:

Figura 5.4: Interfaz de HandVisualizer.

Es un excelente ejemplo de cómo funcionan las dos tecnologías unidas, aprovechando esta información para nuestro proyecto.

5.2.3 Gesturio

Gesturio [16], también programada en JavaScript, esta aplicación nos permite reconocer los gestos estáticos que se encuentran previamente grabados. En este caso son las letras del lenguaje de signos, tanto como el americano como el alfabeto ruso.

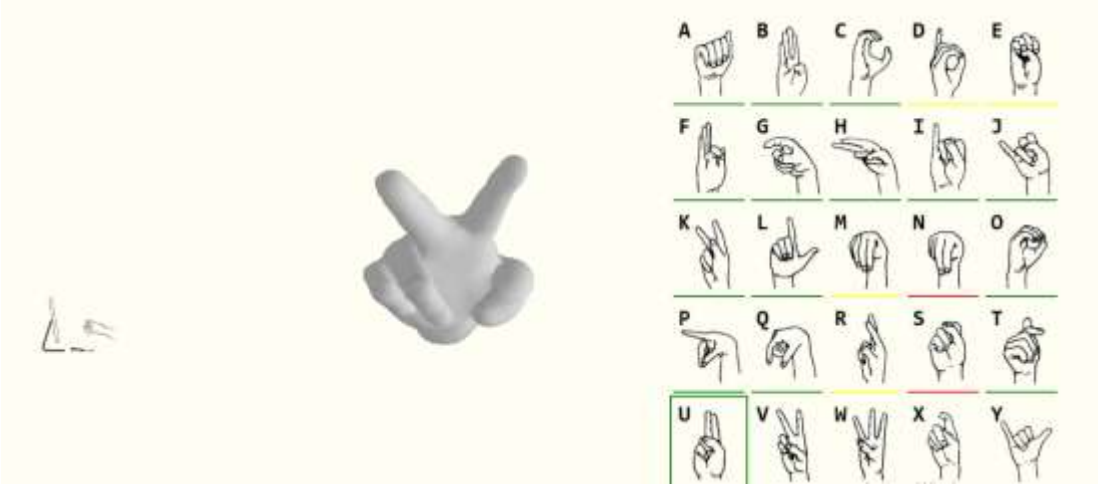


Figura 5.5: Interfaz de Gesturio.

El algoritmo de reconocimiento se basa en la búsqueda del vecino próximo más parecido. Primero transforma el espacio de entrada en espacio de características (Ver figura 5.6).

Esta transformación se basa en el modelo clásico de una mano con 23 grados de libertad. (4 para la orientación de palma y 19 para la posición de los dedos).

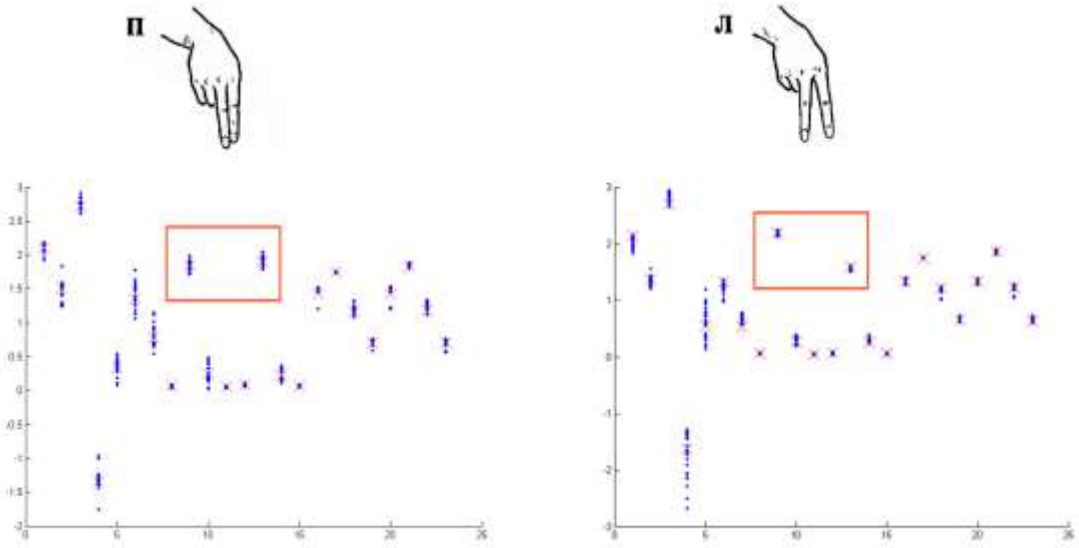


Figura 5.6: Espacio de características de los gestos.

Para encontrar el vecino más cercano se ha usado varios métodos y métricas diferentes, tales como la distancia euclidiana y la distancia L1.

6. SISTEMA DESARROLLADO

6.1 Descripción del sistema:

El proyecto se centra en la creación de un prototipo que permita grabar gestos que luego puedan ser reconocidos mediante el Leap Motion e integrarlos en un entorno 3D. Esto nos permitirá poder definir movimientos con las manos, e integrarlo en cualquier aplicación.

Figura 6.1: Interacción ordenador y Leap Motion.

Un ejemplo básico seria: Bajar y subir el volumen mediante gestos en una aplicación mediante la asociación y grabación de gestos asociados con esta acción, evitando por tanto, la necesidad de incluir un control que requiera de la interacción mediante el uso de un ratón o teclado.

Si una persona tiene un juego o aplicación para pc, y le quiere añadir la interacción mediante el uso de Leap Motion, solo tendrá que crear los gestos necesarios para controlar su juego e integrarlos. Por ejemplo, si la aplicación se controla mediante seis teclas, entonces se tendrán que crear al menos seis gestos. Al hacer un gesto, se ejecutara la acción que previamente se había asociado con la pulsación de una de las teclas.

Este proyecto se hizo para que fuese lo más eficiente posible, ya que un usuario podría querer realizar un gesto y que realice lo más rápido posible la acción, como podemos observar en al siguiente figura:

Figura 6.2: Interacción ordenador y Leap Motion 2.

La aplicación del usuario guardara los frames necesarios para identificar el gesto que usa, para que al realizar un movimiento compare con los gestos almacenados. Si se reconoce el gesto, se realiza la acción que tengamos programada, y dependiendo del gesto, se realiza una acción u otra. En la siguiente figura se explica el procedimiento que se realiza:

Figura 6.3: Lógica usada por el sistema.

6.2 Fases de desarrollo:

El proyecto se desarrolló a lo largo del segundo semestre del curso 2015-2016. Se dividió en 6 fases:

- **Fase 1: Documentación y lectura de la API.**

Esta primera fase consiste en documentarse sobre el Leap Motion para poder analizar su alcance y el uso que se le podría dar. Con ello se podrá definir los requisitos del prototipo y sus funcionalidades mínimas.

- **Fase 2: Preparación del entorno de trabajo.**

El objetivo de esta tarea es instalar y probar el entorno de desarrollo, para luego ejecutar ejemplos básicos. Esto nos es útil para comprobar que el compilador y el

entorno de ejecución estén funcionando correctamente. También incluimos en esta parte el entorno de Ogre, donde también se probaron ejemplos sencillos.

- **Fase 3: Estado de la cuestión.**

Se trata de la realización de un documento que consiste en la búsqueda y análisis de las librerías y programas que usan Leap Motion y Ogre que puedan cumplir con alguno de nuestros requisitos.

- **Fase 4: Definir los gestos y movimientos que se usaran en la aplicación.**

En esta tarea se definirán los gestos y movimientos que grabaremos para su uso en el entorno 3D.

- **Fase 5: Creación del prototipo.**

    - **Especificaciones.**

      Después de haber analizado el alcance del Leap Motion, se tendrán que definir los requisitos y las funcionalidades mínimas del prototipo, de las cuales son obligatorias: Poder grabar y reconocer un gesto.

    - **Diseño.**

      Una de las fases más importantes del desarrollo es la elección de un entorno 3D con el cual podamos trabajar con nuestras especificaciones.

    - **Implementación.**

      Después de saber cuáles serán las especificaciones y el diseño del prototipo, se tendrá que implementar el sistema cumpliendo con los requisitos definidos, siendo esta tarea la más larga.

- **Pruebas.**

  Se harán pruebas a lo largo de la implementación para asegurarse de que el prototipo funciona correctamente.

- **Fase 6: Generación de la documentación.**

  Esta fase se realizó al final del proyecto, cuando estaban todas las funcionalidades del proyecto funcionando correctamente.

6.3 Dificultades y problemas encontrados:

Una de las dificultades del proyecto fue la escasa información de proyectos relacionados con Leap Motion y Ogre, dado que Ogre es más antiguo que Leap Motion.

Para afrontar dichos retos se analizó la documentación existente y manuales sobre la API de Leap Motion y los tutoriales de Ogre.

Cuando aprendí a manejar la API, el objetivo principal fue diseñar e implementar un prototipo en el que se pudieran grabar gestos y reconocerlos, para así ver su veracidad.

Posteriormente aprendí a desarrollar entornos sencillos en Ogre e intente diseñar pequeños prototipos para enlazar Leap Motion con este entorno.

Otra de las mayores dificultades fue encontrar un entorno 3D idóneo para implementar mis gestos, ya que la mayoría eran bastantes complejos para mi conocimiento en Ogre.

Cabe destacar que para el reconocimiento de estos gestos usaríamos el algoritmo DTW (Dynamic Time Warping), que era nuevo para mí.

6.4 Evaluación de requisitos:

Antes de empezar a implementar el prototipo, se establecieron una serie de requisitos para especificar su alcance.

En este apartado se definen las funcionalidades mínimas que tendrá el prototipo, y los requisitos avanzados que se implementaran.

6.4.1 Requisitos básicos.

El prototipo tiene 3 funcionalidades mínimas:

- **Permitir definir gestos:**

  El usuario tendrá que poder grabar un movimiento con su mano. Para ello tendré que recoger los datos del gesto, procesarlos y guardarlos en un formato específico.

- **Reconocer un gesto:**

  El prototipo tendrá que reconocer si el usuario hace un movimiento similar a otro ya grabado. Para ello, usare el programa el algoritmo DTW para comparar en cada instante el gesto que se está realizando con los gestos grabados anteriormente.

6.4.2 Requisitos avanzados.

Además de las funcionalidades mínimas, se establecieron otros requisitos avanzados para hacer el prototipo más eficiente:

- **Gestos en tiempo real:**

  Debido a que nos movemos dentro de un juego, un factor crítico es el tiempo, se tiene que optimizar el algoritmo y el juego para que los movimientos sean lo más parecidos posibles al movimiento de la mano.

- **Implementar entorno 3D**

  Crear un pequeño entorno 3D para implementar esos gestos.

6.5 Datos que se obtienen del Leap Motion:

Para obtener datos del Leap Motion, tenemos que crear un escuchador, este dispositivo nos devuelva una serie de imágenes instantáneas que detecta en su campo de emisión, **frames**. El número de *frames* por segundo que devuelve el dispositivo depende de los recursos de la máquina, el número de elementos en su campo de visión, los ajustes de seguimiento y otros factores.

Se recomienda que el dispositivo se sitúe a unos 30 cm, en frente del monitor. Lo más centrado posible. Podemos hacernos una idea de su rango de visión observando la figura 6.4.

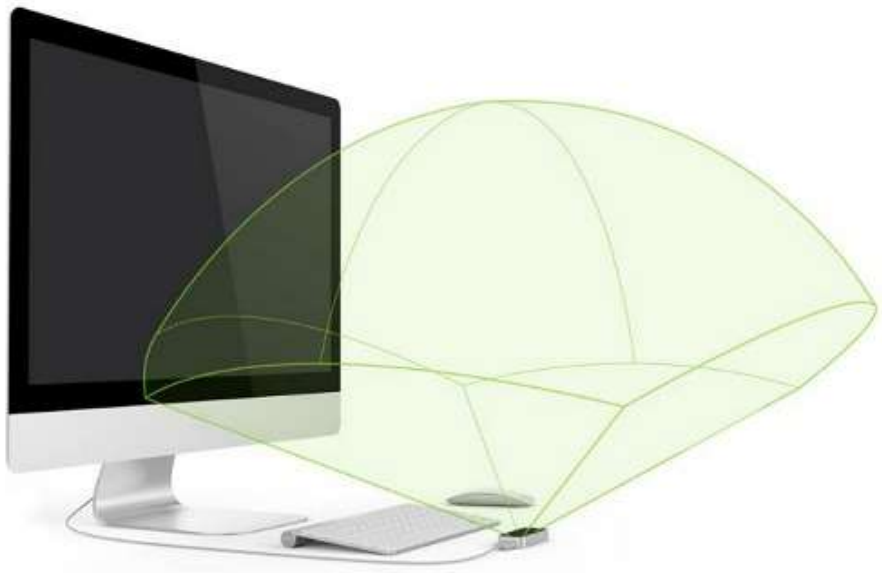


Figura 6.4: Área de detección.

Para el correcto funcionamiento del juego se necesita una batería de gestos predefinidos que se deben grabar para que el reconocedor pueda compararlos y reconocerlos correctamente permitiendo la continuación de la ejecución del juego.

Debido a que queremos un juego en tiempo real, debemos escoger justo los datos necesarios para que el algoritmo no se demore, en el apartado de pruebas estudiaremos los diferentes gestos a realizar. Hemos clasificado los gestos en dos tipos:

6.5.1 Gestos básicos:

Para controlar el entorno 3D debemos definir gestos intuitivos para que el usuario no tenga dificultad en realizarlos y puedan entenderlos. Aquí vamos a explicar que datos recogemos para los gestos básicos.

- La dirección de donde apuntan la mano:

Figura 6.5: Dirección hacia done apunta la mano.

Como observamos en la figura 6.5, la dirección se expresa como un vector unitario en la misma dirección en la que apuntan los dedos. De esta manera podemos identificar si estamos indicando movimientos hacia derecha, izquierda o hacia delante.

- El vector normal a la palma:

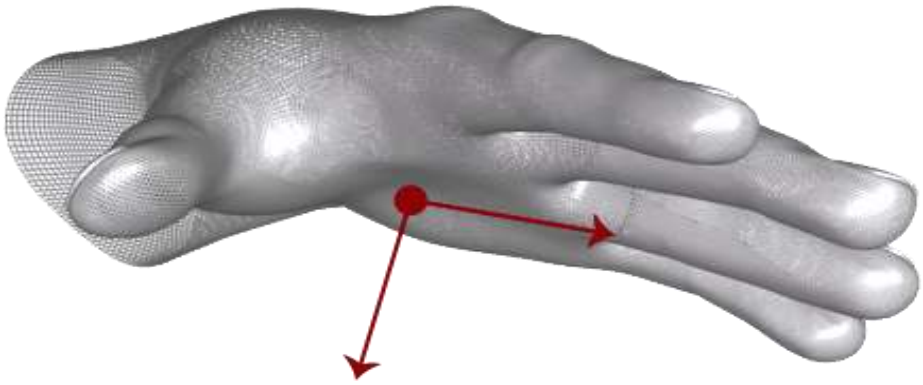


Figura 6.6: Vector normal de la palma de la mano.

La dirección se expresa como un vector unitario en la misma dirección que la palma de la mano (es decir, un vector ortogonal a la palma de la mano) (Ver figura 6.6). Con esto podemos observar el ángulo de elevación de la mano.

6.5.2 Gestos avanzados:

Debido a que la API nos ofrece mayores posibilidades de reconocimiento se han definido las siguientes características para reconocer gestos más avanzados como, por ejemplo, las letras del lenguaje de sordos [18], para ello hemos obtenido:

- Dirección hacia a donde apunta cada dedo:

Figura 6.7: Dirección hacia donde apunta cada dedo de la mano.

En este caso realizamos algo más exacto que en el apartado anterior. Primero de la mano obtenemos una lista de todos los dedos que detectamos y posteriormente recogemos hacia donde apunta cada uno. De esta manera podemos diferenciar bastantes gestos diferentes solo poniendo los dedos en posiciones diferentes.

- Angulo entre la palma de la mano y los dedos

  Se trata del ángulo que forma la palma de la mano y los dedos. Se calcula sin tener en cuenta el dedo pulgar. Obtenemos 0 radianes para el caso de la mano abierta hasta pi radianes cuando es un puño apretado.

  Con esto podemos obtener lo cerrada que esta la mano y diferenciar posiciones diferentes.

6.6 Gestor de gestos:

Debido a que este juego pueden acceder varios usuarios, vamos a crear un gestor de gestos para almacenar gestos y utilizarlos por varios usuarios, o por el contrario, si se quieren volver a grabar nuevos gestos.
Para la grabación e identificación de gestos en pruebas, hemos desarrollado una aplicación de consola, donde encontramos el siguiente menú:

Figura 6.8: Menú principal OgreTank.

6.6.1 Entrenar gestos:

Para asignar una acción a cada tipo de gesto, tendremos que grabarla, para ello se creó esta función que nos permite grabar los gestos y guardarlos en un archivo. Estos datos los usaremos después para su reconocimiento.

Si pulsamos esa opción nos aparecerá la siguiente pantalla:

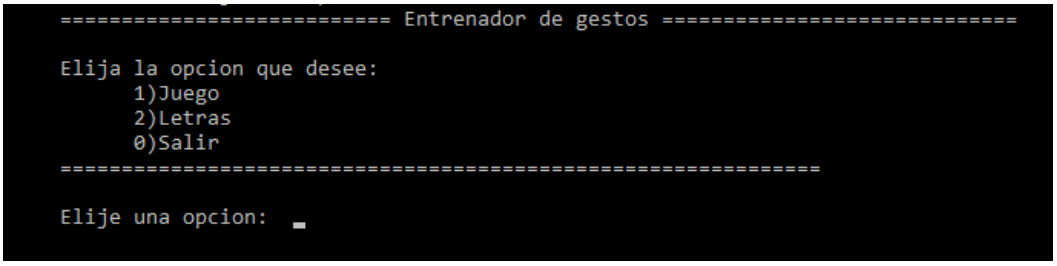


Figura 6.9: Entrenador gestos OgreTank.

Hemos implementado dos tipos de entrenamiento, el primero con los gestos necesarios para controlar nuestro juego y por otra parte las cinco primeras letras del abecedario, en lenguaje de signos, para así comprobar la calidad de nuestro sistema.

Como podemos ver en estas imágenes:

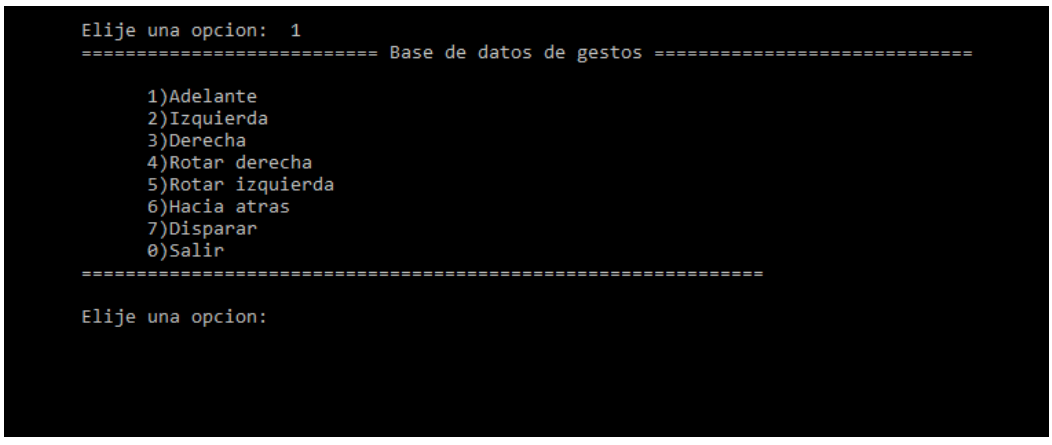


Figura 6.10: Base de datos gestos juego.

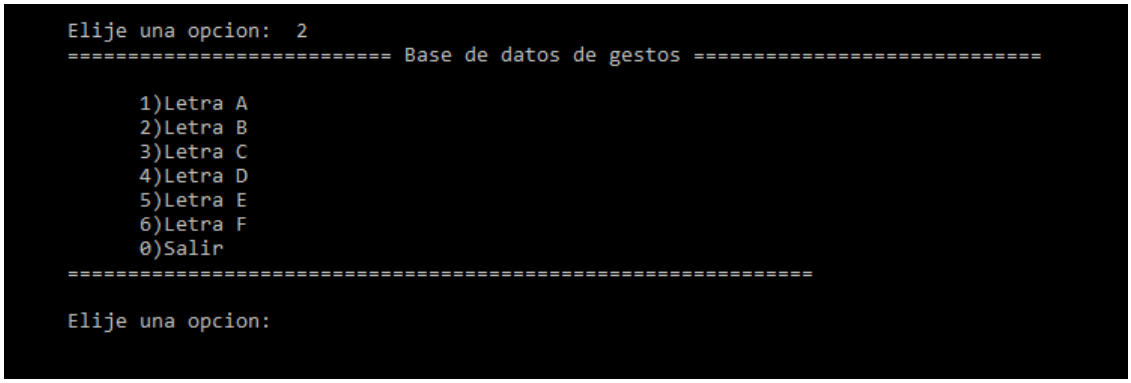


Figura 6.11: Base de datos gestos letras.

Cuando el usuario lanza la grabación, se activa el escuchador del Leap Motion para la recepción de datos. En cada *frame* guardamos los datos explicados anteriormente en el apartado 6.6.

Para cada gesto que vayamos grabando, nos ira mostrando el porcentaje de datos (Figura 6.12) que ha recogido para completar el entrenamiento. En nuestro caso se nos repite tres veces debido a que hemos decidido que por defecto nos grabe tres repeticiones para cada gesto.

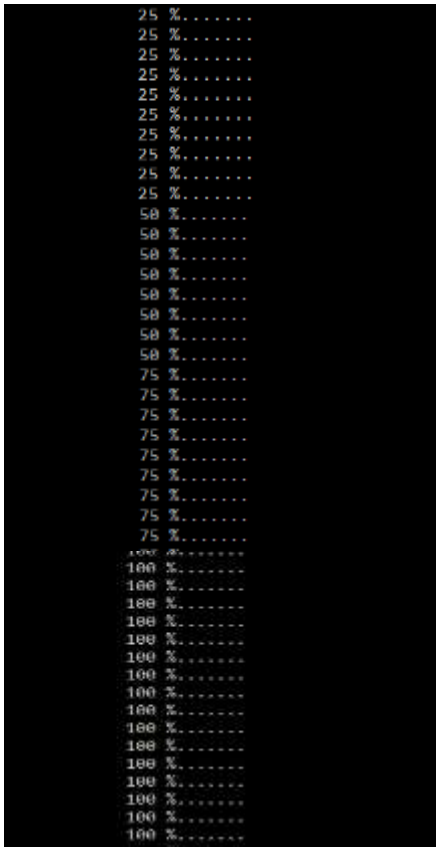


Figura 6.12: Porcentaje de grabación gestos.

Una vez completado el reconocimiento del gesto volveremos al menú principal, donde volveremos para elegir el gesto siguiente a grabar, o si por el contrario hemos acabado salir del entrenador de gestos.

6.6.2 Tiempo real:

Debido a que este gestor esta realizado para realizar pruebas y comprobar los fallos y aciertos del prototipo, con esta función podemos ir probando gestos y nos devolverá el gesto calculado en tiempo real.

Antes de ejecutar esta opción, ya hemos elegido anteriormente cargar los datos del Juego o por el contrario los datos de las Letras, con lo cual el sistema ya automáticamente muestra los resultados que detecte de los gestos cargados, como podemos ver:

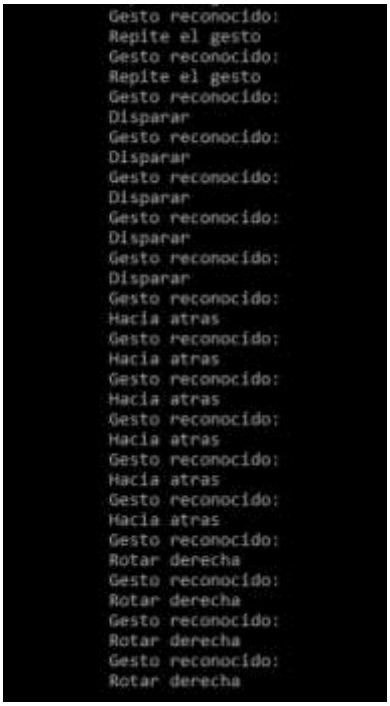


Figura 6.13: Resultados gestos predichos.

En este caso, hemos cargado los gestos del juego.

6.6.3 Entrenar sistema

Una parte importante del sistema es el entrenamiento, con esta opción cargaremos la base de datos de los gestos grabados por el usuario y seguidamente entrenaremos nuestro sistema. Generaremos el modelo particular del usuario para que el sistema sea personalizado.

De nuevo podemos elegir si el entrenamiento es para gestos del juego o por el contrario las letras antes mencionadas.

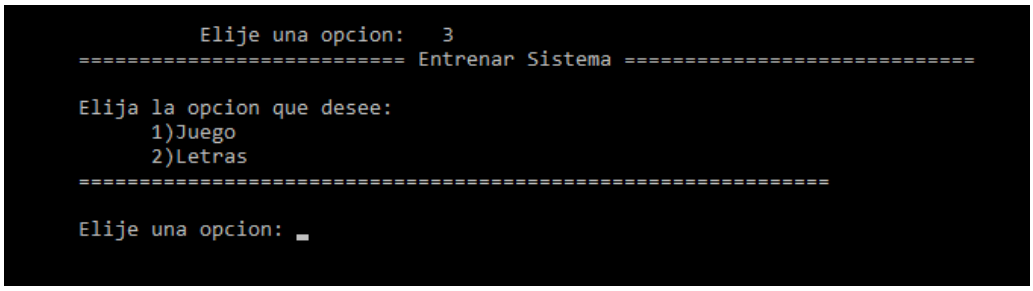


Figura 6.14: Entrenar sistema.

6.6.4 Cargar entrenamiento por defecto:

Con esta función cargaremos los datos que hemos grabado por defecto para nuestros gestos y así poder probar la veracidad de estos en el apartado de pruebas.

6.6.5 Ejecutar juego:

Inicializamos el juego, para ello hay que destacar que habría que entrenar antes el sistema, si se quiere personalizado, por el contrario cargara por defecto un entrenamiento.

6.7 Grabador de gestos:

Como hemos mencionado en el apartado anterior, para la grabación de gestos elegimos la opción → 1-Entrenar gestos.

Hemos configurado para añadir tres muestras de cada gesto, con lo cual debemos realizar el gesto tres veces seguidas.

Para ello, primero inicializamos un escuchador o listener, de Leap Motion, para poner a escuchar el dispositivo y recoger todos los datos necesarios.

A la hora de iniciar la grabación, tiene que detectar al menos una mano sobre el dispositivo, sino no iniciaría la grabación de datos.
Se trata de un proceso iterativo, donde un contador, va rellenando vectores con la información requerida según el tipo de gesto a realizar.

Cuando tenemos todos los datos, los copiamos en un vector para la introducción de estos en el algoritmo DTW. Este vector será del tamaño que hayamos definido en los vectores de datos.

Una vez finalizado la copia de datos, guardamos el documento con todos la base de datos en formato .txt. Normalmente con el nombre DTWTraining.txt.

El formato de DTWTraining.txt es el siguiente:

```
GRT_LABELLED_TIME_SERIES_CLASSIFICATION_DATA_FILE_V1.0
DatasetName: Data
InfoText: This data contains some timeseries data
NumDimensions: 6
TotalNumTrainingExamples: 25
NumberOfClasses: 5
ClassIDsAndCounters:
1       5
2       5
3       5
4       5
5       5
UseExternalRanges: 0
LabelledTimeSeriesTrainingData:
************TIME_SERIES************
ClassID: 1
TimeSeriesLength: 143
TimeSeriesData:
-0.209405       -0.140356       -0.967703       -0.0319357      -0.999428       0.011085
-0.209327       -0.141085       -0.967614       -0.0317816      -0.999439       0.0105906
-0.209418       -0.141774       -0.967494       -0.0316645      -0.999446       0.0102285
-0.209565       -0.141616       -0.967485       -0.0315911      -0.999451       0.0100119
-0.209404       -0.141268       -0.967571       -0.0315814      -0.999456       0.0095102
-0.209261       -0.141164       -0.967617       -0.0314876      -0.999465       0.00887205
-0.209041       -0.141153       -0.967666       -0.0313958      -0.999473       0.00822116
-0.208907       -0.140701       -0.967761       -0.0313567      -0.999479       0.00768424
-0.208753       -0.140518       -0.967821       -0.0313747      -0.999482       0.00719832
-0.208454       -0.140437       -0.967897       -0.0313618      -0.999486       0.00665152
-0.208146       -0.140084       -0.968014       -0.0313813      -0.999489       0.00612585
-0.20805        -0.139754       -0.968082       -0.0313437      -0.999492       0.00575336
-0.208002       -0.139514       -0.968128       -0.0312979      -0.999495       0.00540986
-0.207942       -0.138955       -0.968221       -0.0312742      -0.999498       0.00503938
-0.207697       -0.139183       -0.968241       -0.0312433      -0.999501       0.00471145
-0.207542       -0.138957       -0.968306       -0.0312487      -0.999502       0.00438072
-0.207446       -0.138291       -0.968422       -0.0312355      -0.999505       0.00368382
-0.207169       -0.137987       -0.968525       -0.0313146      -0.999505       0.00290073
```

Figura 6.15: Datos recogidos en algoritmo DTW.

Donde podemos destacar los siguientes campos:

- NumDimensions: Numero de columnas de datos que contiene cada gesto.
- TotalNumTrainingExamples: número total de datos de gestos que hemos introducido (Hemos definido realizar el gesto 5 veces).
- NumberOfClasses: número de gestos que hay en la base de datos.
- ClassID: numero identificador del gesto.
- TimeSeriesLength: cantidad de datos que hay en ese gesto

6.8 Entrenar sistema:

Con esta opción cargamos los archivos por defecto creados, realiza las siguientes acciones:

- Cargamos el archivo DTWTraining.txt
- Entrenamos el gestor de gestos, esto nos creara un modelo llamado DTWModel.txt.

Este archivo es el que usa el algoritmo DTW para ir comparando los gestos que hemos guardado en la base de datos, después de definir el umbral el sistema nos indicara cual es el gesto que ha predicho. El archivo tiene el siguiente formato:

```
GRT_DTW_Model_File_V2.0
Trained: 1
UseScaling: 0
NumInputDimensions: 6
NumOutputDimensions: 1
NumTrainingIterationsToConverge: 3435973836
MinNumEpochs: 0
MaxNumEpochs: 100
ValidationSetSize: 20
LearningRate: 0.1
MinChange: 1e-005
UseValidationSet: 0
RandomiseTrainingOrder: 1
UseNullRejection: 0
ClassifierMode: 1
NullRejectionCoeff: 3
NumClasses: 5
NullRejectionThresholds:  0 0 0 0 0 0
ClassLabels:  1 2 3 4 5
DistanceMethod: 1
UseSmoothing: 0
SmoothingFactor: 5
UseZNormalisation: 0
OffsetUsingFirstSample: 0
ConstrainWarpingPath: 1
Radius: 0.2
RejectionMode: 0
NumberOfTemplates: 6
OverallAverageTemplateLength: 118
***************TEMPLATE***************
Template: 1
ClassLabel: 1
TimeSeriesLength: 80
TemplateThreshold: 3.06215
TrainingMu: 1.09136
TrainingSigma: 0.656929
```

Figura 6.16: Modelo usado por el algoritmo DTW.

Podemos destacar:

- NumTrainingIterationsToConverge: número de iteraciones que realiza el algoritmo para sacar el modelo.
- DistanceMethod: método para calcular la distancia, en este caso usa la distancia euclidea.
- TemplateThreshold: umbral para el cálculo del gesto, cálculo del umbral explicado en la introducción al algoritmo DTW.
- TrainingMu: umbral de clasificación para la plantilla
- TrainingSigma: distancia deformación total normalizada promedio.

- AverageTemplateLength: la longitud media de los datos entre los recogidos del archivo de entrenamiento.
- UseNullRejection: activar esta función nos permitirá discriminar aquellos gestos que no estén dentro de nuestra base de datos.
- NullRejectionCoeff: se trata del coeficiente de multiplicación del umbral para discriminar los gestos, para evitar falsos positivos.

6.9 El reconocimiento:

Una de las partes más complejas del proyecto es el reconocedor, ya que tiene que ir comparando el gesto que se está realizando en tiempo real con los diferentes movimientos cargados.

6.9.1 El algoritmo:

Como hemos mencionado anteriormente, usaremos el algoritmo DTW, para la comparación de los diferentes gestos.

A continuación, explicare un ejemplo para entender cómo podríamos usar dicho algoritmo con los datos del Leap Motion [11]. Tenemos guardado el movimiento de una mano, lo que significa que tenemos una lista de posiciones en el eje X Y Z que lo definen. Dichos datos están guardados en el archivo DTWTrainingData.txt.
Este ejemplo demuestra cómo entrenar y utilizar el algoritmo DTW para la clasificación. El ejemplo carga los datos mostrados en la imagen de abajo y la utiliza para entrenar el algoritmo DTW. Este conjunto de datos contiene 3 series temporales gestos de ratón, que escriben las letras G, R y T. G es la clase 1, R es la clase 2, y T es la clase 3.

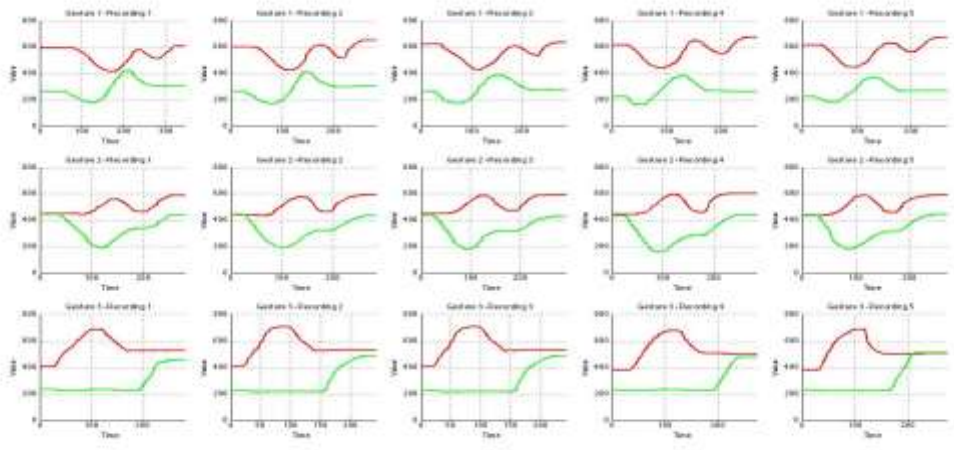


Figura 6.16: Ejemplos DTW.

Cuando se lanza el reconocedor, se tendrá que ir comparando los gestos realizados con los gestos cargados en tiempo real. Hemos definido que se recojan el 75 % de los datos posibles, para así hacer más eficiente el algoritmo.
Para hacernos una idea, en esta parte explicaré de forma muy breve los 2 elementos más importantes que forman el reconocedor:

- **El escuchador:**

  El escuchador es la parte que recibe los datos del Leap Motion y que los introduce en la hoja de datos. Después de añadir el nuevo dato al vector para crear la hoja de datos, el escuchador se encargará de introducirlo. Después de copiar los datos se eliminaran los datos almacenados en los vectores para no solapar datos.

- **El comparador:**

  Después de tener estas series temporales entrenamos el algoritmo, que nos devolverá el archivo DTWModel.txt mencionado anteriormente. Con este modelo vamos comparando el gesto que vayamos grabando en tiempo real y viendo con cual tiene mayor similitud. El algoritmo nos devolverá la clase con la que puede coincidir.

La lógica usada por el sistema se puede visualizar en la siguiente figura:

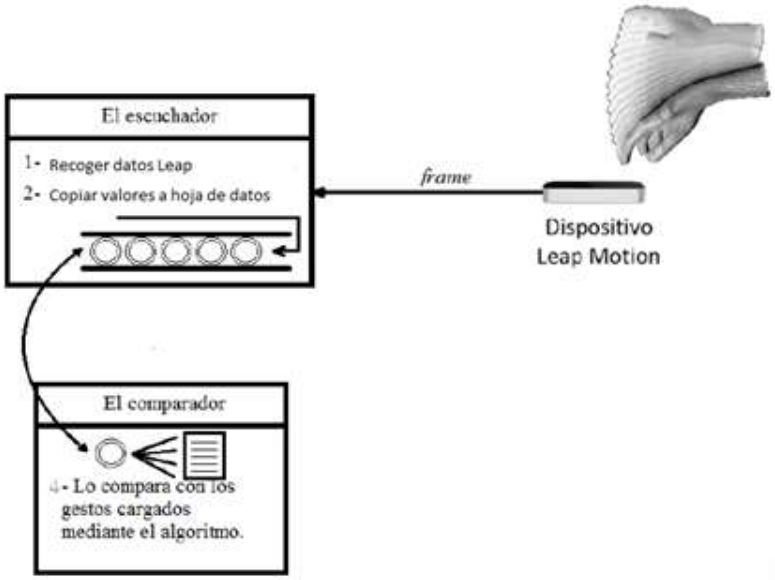


Figura 6.17: Proceso completo.

6.9 Integración con entorno 3D

En este apartado vamos a explicar la integración del reconocedor y grabador de gestos realizado en apartados anteriores a un entorno 3D. Después de una larga búsqueda de un entorno 3D que se ajustara a mi conocimiento sobre Ogre decidí elegir el proyecto denominado OgreTank.

6.9.1 Introducción de OgreTank:

OgreTank [17] es un proyecto desarrollado por Wei Fang y Yi Xu [ ], se trata de un juego realizado en Ogre en el cual controlamos un tanque. En el cual de manera aleatoria aparecen nuevos tanques que van desplazándose y disparando. También de manera aleatorio nos encontraremos paquetes especiales, si los obtenemos podemos adquirir algunas características especiales como inmunidad o disparar tres balas a la vez. El objetivo de este juego es eliminar a los demás tanques, para ello habrá que controlar nuestro tanque, ahí es donde aplicamos nuestra aplicación.

Gracias al reconocedor de gestos que implementamos, podemos manejar el tanque y jugar sin tocar el teclado. Por defecto el tanque se puede controlar por teclado, pero nosotros cambiaremos esas opciones.
Elegí este proyecto debido a que es un entorno sencillo aunque tenga algunas partes de diseño 3D, como son las colisiones o disparos, la parte de movimientos del tanque podía realizarlas con mis gestos.

Algunas capturas de pantalla del entorno con algunas de los paquetes especiales que he mencionado anteriormente:

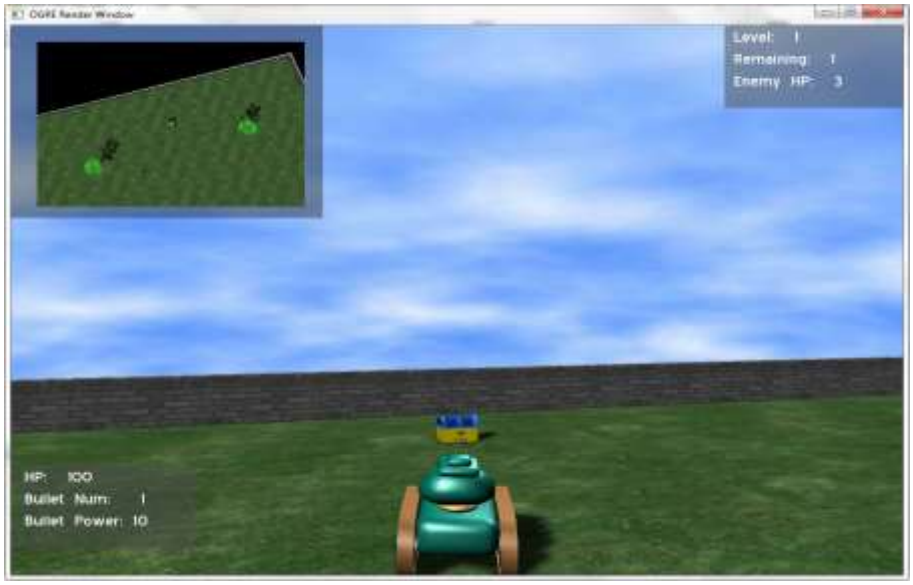


Figura 6.18: Entorno 3D, donde se puede apreciar uno de los objetos que se pueden conseguir.

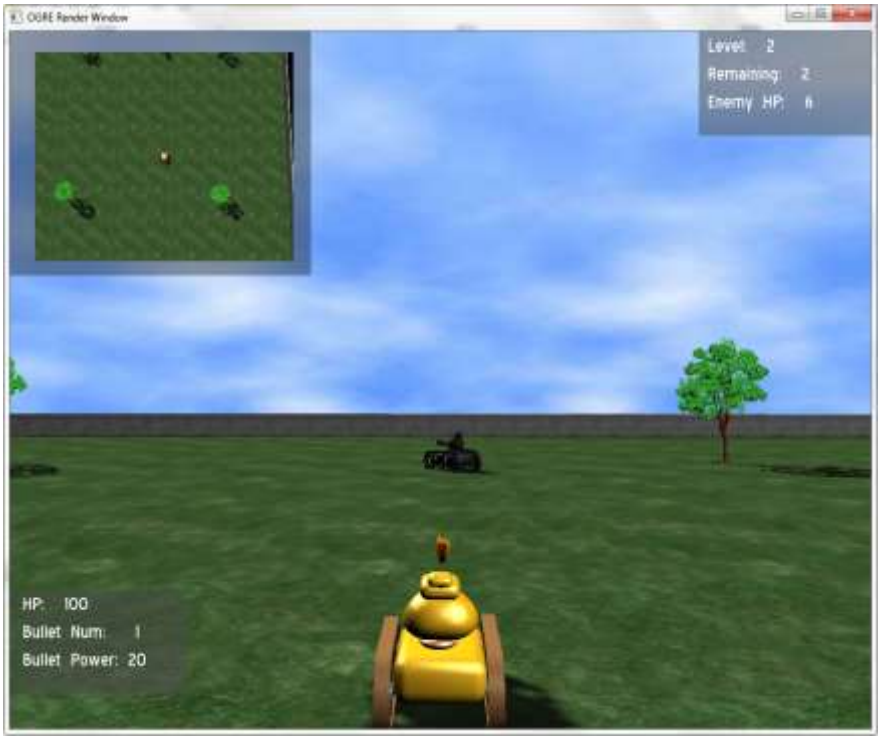


Figura 6.19: Entorno 3D, se muestra en estado de inmunidad, debido a unos de los paquetes de bonus que se pueden adquirir.

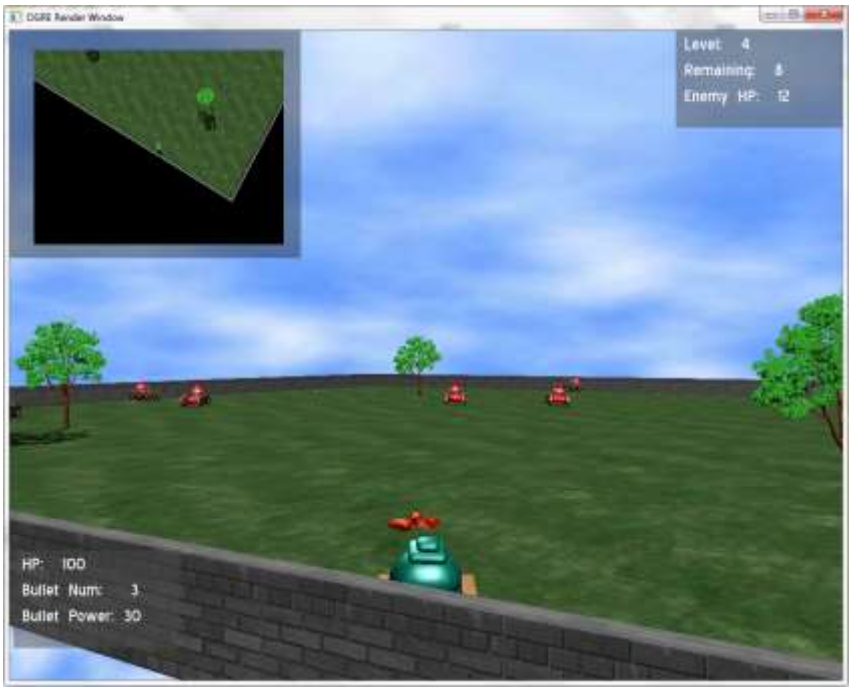


Figura 6.20: Entorno 3D, una nueva modificación por paquete, donde puede disparar tres proyectiles al mismo tiempo.

6.9.2 Movimientos realizados por el tanque:

En este apartado introduciremos los movimientos que podemos realizar dentro del entorno para especificar qué tipo de gestos podemos realizar para que sean intuitivos para el usuario.

El entorno por defectos tiene implementado los siguientes movimientos:

- Moverse hacia delante.
- Moverse hacia atrás.
- Rotar hacia la izquierda.
- Rotar hacia la derecha.
- Rotar el cañón hacia la derecha.
- Rotar el cañón hacia la izquierda.
- Disparar

Vamos a definir siete gestos para la realización de acciones predefinidas con diferentes ID.
Hemos asignado las ID a las siguientes acciones:

- Moverse hacia delante → ID 1
- Rotar hacia la izquierda → ID2
- Rotar hacia la derecha → ID3
- Rotar el cañón hacia la derecha → ID4
- Rotar el cañón hacia la izquierda → ID5
- Disparar → ID6
- Moverse hacia atrás → ID7

6.9.3 Implementación gestos en OGRE3D:

Para la inserción en el juego interactivo ha habido que realizar algunas funciones para grabar, reconocer gesto y realizar la acción definida.
En primer lugar se tuvo que extraer el código del reconocedor de gestos con el algoritmo de DTW e insertarlo en el proyecto. Una de las opciones que teníamos en el

gestor de gestos era la de cargar los datos anteriores, para este caso se tendría que cargar automáticamente para que podamos usar el juego al iniciarse Ogre.

Para ello hemos creado una hebra, en el cual llamamos a la misma función cargar del reconocedor, con lo cual a la misma vez que iniciamos el juego empieza a cargar los datos predefinidos. Después de obtener dichos datos esa hebra será eliminada.

Una vez cargado los gestos e iniciado el entorno 3D, creamos un escuchador para recoger todos los frames que nos devuelva Leap Motion. Volvemos a recordar que por defecto nos devolverá un gesto cuando gestione el 75% de muestras recibidas por el Leap Motion, debido a que es lo más eficiente para nosotros.

Cuando obtengamos los suficientes frames para detectar un gesto creamos la segunda hebra, en esta copiaremos los frames recogidos para poder comparar el gesto con el modelo creado por el algoritmo DTW, devolveríamos una ID con la que estaría relacionada una acción.

Habiendo definido lo anterior, ahora es el momento de modificar el código del entorno 3D, nos dirigimos a la clase *UserInput*, y dentro de esta modificamos la función *Think*.

Lo primero que debemos definir en esta función es lo siguiente, si no detectamos ninguna mano sobre el Leap Motion no se realice ninguna acción para evitar falsos positivos.
Asignamos cada ID con una acción del tanque, como hemos explicado en el punto anterior.

También hemos forzado el sistema a que no empiece a recoger frames hasta que no haya devuelto un gesto, así evitamos la perdida de frames mientras realizamos el gesto, aunque es casi instantáneo.
Debido al elevado gasto de computación de cálculo del algoritmo DTW y el renderizado del entorno 3D no hemos conseguido hacer que el movimiento sea exactamente a la misma vez que realizamos el gesto.
En resumen estas son las acciones que realiza nuestro prototipo:

- Cargar entorno 3D y archivos donde se encuentra la base de datos de gestos.
- Habilitar escuchador y recoger frames necesarios para realizar gestos.
- Calcular con DTW el gesto realizado y devolver ID calculado.
- Renderizar el entorno 3D y realizar la acción asociada

El sistema se basa en la multihebra, para que funcione lo más fluido posible.

7.PRUEBAS Y RESULTADOS

Para analizar el prototipo, se hicieron una serie de pruebas. Definimos una lista de tareas que realizaran una serie de personas para probar el funcionamiento del prototipo.

Antes de empezar la realización de las pruebas, es necesario explicar a los usuarios las distintas funcionalidades y objetivos del prototipo.

7.1. Prueba 1: Grabación.

La primera tarea consiste en grabar nuevos gestos. Los usuarios no tardaron en localizar el botón que tiene la opción de entrenar gestos, dado que en la ventana principal se encuentran la mayoría de las funcionalidades importantes.

7.2. Prueba 2: Cargar.

La segunda tarea consiste en cargar el archivo TXT que contiene los nuevos gestos que el usuario ha grabado en la prueba anterior en el reconocedor o que están definidos por defecto.
Después de cargarlos recibirán un mensaje de confirmación de que se han cargado.

7.3. Prueba 3: Reconocimiento.

La última tarea consiste en lanzar el reconocedor y realizar los diferentes gestos grabados en la prueba 1. De esta forma el prototipo reconocerá los gestos.

7.4 Resultados:

El proceso que se ha seguido en las pruebas para obtener los resultados ha sido el siguiente:

- Se ha explicado el manual de usuario.
- El usuario ha grabado los gestos que se le indican.
    - Han comprobado por si mismos el resultado de su entrenamiento.
- El usuario ha cargado por defecto el archivo de entrenamiento
    - Han comprobado por si mismos el resultado de su entrenamiento.

Al realizar las pruebas con los usuarios, pude observar diferentes situaciones, de las cuales las más importantes:

- Cuando un usuario se equivocaba en rellenar los datos, no sabía cómo parar la grabación y reiniciarla.

- La segunda prueba no tenía dificultad, y los usuarios consiguieron cargar los movimientos anteriormente grabados para ser reconocidos.

- El hacer la interfaz en consola, a la mayoría de personas le asustaba pulsar la opción que era errónea, sería una buena opción para mejorar.

7.4.1 Pruebas gestos sencillos:

- Avanzar:

Figura 7.1: Gesto avanzar.

- Girar izquierda:

Figura 7.2: Gesto girar tanque izquierda.

- Girar derecha:

Figura 7.3: Gesto girar tanque derecha.

- Rotar izquierda:

Figura 7.4: Gesto rotar cañón izquierda.

- Rotar derecha:

Figura 7.5: Gesto rotar cañón derecha.

Persona 1:

	Correcto	Falso Positivo	Fuera de umbral	Aciertos
Adelante	10	0	0	100 %
Izquierda	9	1	0	90%
Derecha	10	0	0	100%

Rotar izq	9	1	0	90%
Rotar der	8	2	0	80%

Tabla 1: Prueba persona 1.

Persona 2:

	Correcto	Falso Positivo	Fuera de umbral	Acierto
Adelante	10	0	0	100%
Izquierda	10	0	0	100%
Derecha	10	0	0	100%
Rotar izq	8	2	0	80%
Rotar der	9	0	1	90%

Tabla 2: Prueba persona 2.

Hemos observado que el acierto es casi total en los gestos sencillos, con lo cual hemos decidido solo hacer 2 pruebas.

7.4.2 Pruebas gestos avanzados:

Para estas pruebas se pensó colocar Leap Motion en la siguiente posición:

Figura 7.6: Prueba posición de Leap Motion.

Hemos tenido que desestimar esta opción debido a que, como podemos observar en la siguiente imagen, vemos que detecta una mano aunque no la haya. Esto es debido a que como explicamos anteriormente cuando ilumina la zona con luces infrarrojas observamos que nuestro cuerpo refleja esta luz y nos detecta una falsa mano.

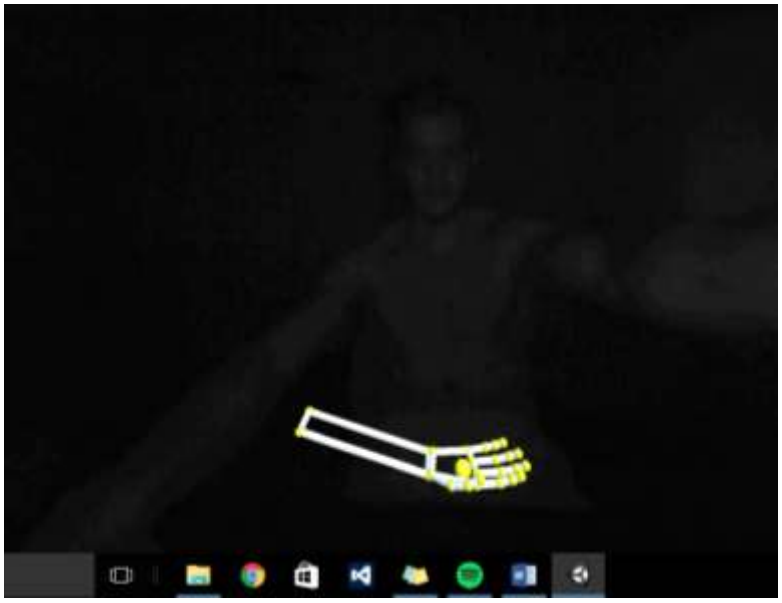


Figura 7.7: Visión Leap Motion.

Las pruebas se han realizado con el dispositivo en la siguiente posición:

- Letra A:

Figura 7.8: Gesto letra A.

- Letra B:

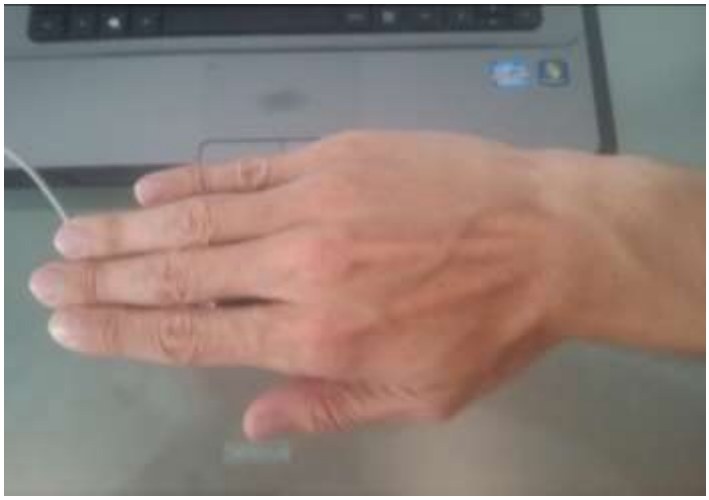


Figura 7.9: Gesto letra B.

- Letra C:

Figura 7.10: Gesto letra C.

- Letra D:

Figura 7.11: Gesto letra D.

- Letra E:

Figura 7.12: Gesto letra E.

- Letra F:

Figura 7.13: Gesto letra F.

Para este apartado hemos decidido hacer dos tipos de comprobaciones, en el primero lo que haremos será generalizar el entrenamiento, es decir, hacer un solo entrenamiento y probarlo. Seguidamente haremos una segunda comprobación, para ello cada sujeto entrenara el sistema y comprobara el grado de acierto del sistema.

Estas dos tipos de comprobaciones, las realizaremos para analizar si el juego puede utilizarse sin ningún entrenamiento específico o si por el contrario, necesitaríamos hacer un entrenamiento específico para cada usuario.

Para realizar las pruebas de resultados, se le ha pedido al usuario que realice consecutivamente diez veces el gesto entrenado. En el caso de los gestos avanzados, por ejemplo, realizar diez veces el gesto de la letra A, B, C, D, E y F.

Los resultados son los siguientes:

- **Generalizar.**

  Se carga el archivo de entrenamiento por defecto y se ejecuta la función de pruebas.

Persona 1:

	Correcto	Falsos Positivos	Fuera de umbral	Aciertos
Letra A	8	0	2	80%
Letra B	10	0	0	100%
Letra C	10	0	0	100%
Letra D	6	2	2	60%
Letra E	10	0	0	100%
Letra F	10	0	0	100%

Tabla 3: Prueba generalizar persona 1.

Persona 2:

	Correcto	Falso Positivo	Fuera de umbral	Aciertos
Letra A	2	0	8	20%
Letra B	10	0	0	100%
Letra C	10	0	0	100%
Letra D	1	0	9	10%
Letra E	10	0	0	100%
Letra F	10	0	0	100%

Tabla 4: Prueba generalizar persona 2.

Persona 3:

	Correcto	Falso Positivo	Fuera de umbral	Aciertos
Letra A	3	1	6	30%
Letra B	10	0	0	100%
Letra C	10	0	0	100%
Letra D	2	0	8	20%
Letra E	8	0	2	80%

Letra F	10	10	10	100%

Tabla 5: Prueba generalizar persona 3.

Persona 4:

	Correcto	Falso Positivo	Fuera de umbral	Aciertos
Letra A	3	0	7	30%
Letra B	6	4	0	60%
Letra C	9	0	1	90%
Letra D	1	2	7	10%
Letra E	10	0	0	100%
Letra F	10	0	0	100%

Tabla 6: Prueba generalizar persona 4.

- **Entrenamiento personalizado**:

  El propio usuario entrena al sistema y consecutivamente realiza las pruebas con su base de datos personalizada.

Persona 5:

	Correcto	Falso Positivo	Fuera de umbral	Aciertos
Letra A	10	0	0	100%
Letra B	10	0	0	100%
Letra C	10	0	0	100%
Letra D	10	0	0	100%
Letra E	10	0	0	100%
Letra F	8	0	2	80%

Tabla 7: Prueba personalizada persona 5.

Persona 6:

	Correcto	Falso Positivo	Fuera de umbral	Aciertos
Letra A	10	0	0	100%

Letra B	10	0	0	100%
Letra C	10	0	0	100%
Letra D	8	2	0	80%
Letra E	10	0	0	100%
Letra F	10	0	0	100%

Tabla 8: Prueba personalizada persona 6.

Persona 7:

	Correcto	Falso Positivo	Fuera de umbral	Aciertos
Letra A	10	0	0	100%
Letra B	10	0	0	100%
Letra C	10	0	0	100%
Letra D	5	4	1	50%
Letra E	10	0	0	100%
Letra F	10	10	0	100%

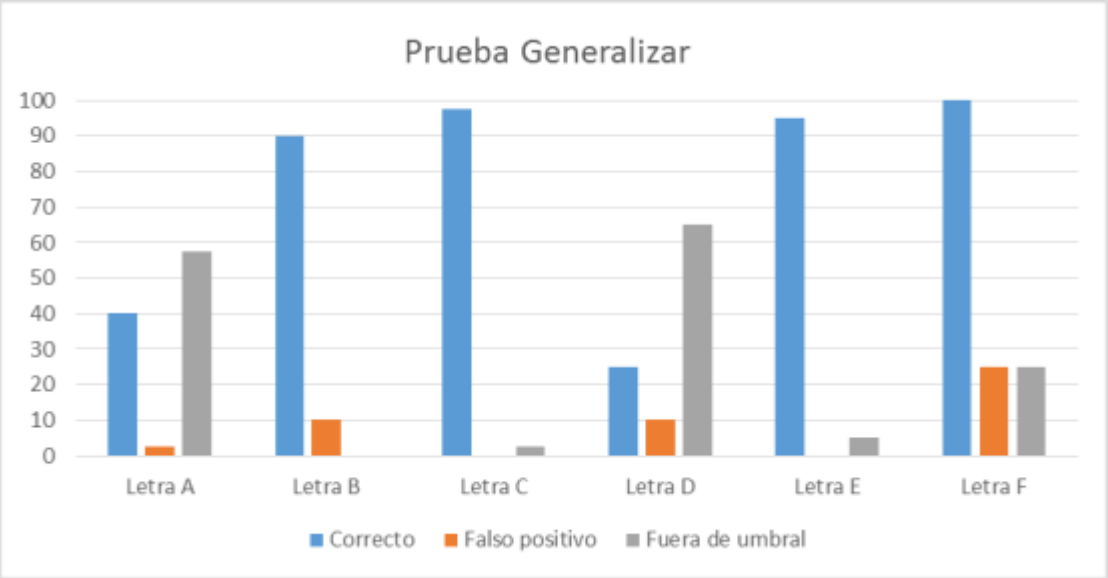
Tabla 9: Prueba personalizada persona 7.

Persona 8:

	Correcto	Falso Positivo	Fuera de umbral	Aciertos
Letra A	3	0	7	30%
Letra B	10	0	0	100%
Letra C	10	0	0	100%
Letra D	0	8	2	0%
Letra E	8	2	0	80%
Letra F	10	0	0	100%

Tabla 10: Prueba personalizada persona 8.

A partir de las siguientes tablas hemos obtenido las siguientes gráficas para sacar nuestras conclusiones:



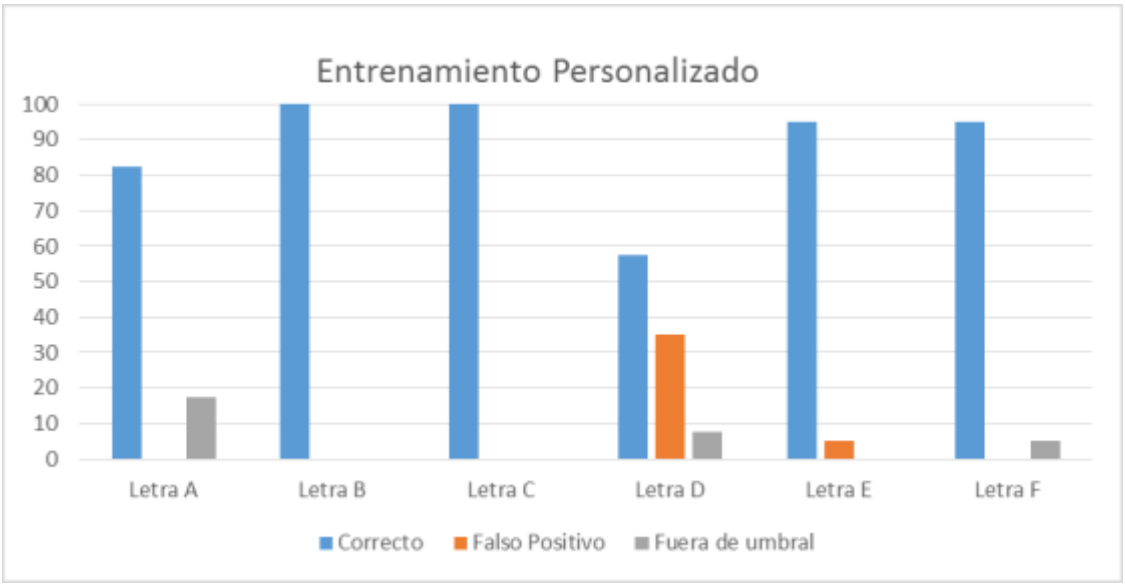
Grafica 1: Resultados entrenamiento generalizado.

Podemos observar que tenemos muy pocos falsos positivos, sin embargo tenemos demasiados gestos no reconocidos, sobre todo en las letras A y D. Puedo suponer que esto se debería a que la letra D algunas veces no detecta correctamente el dedo índice, como podemos ver en esta imagen.

Figura 6.14: Detección errónea letra D.

Figura 6.15: Detección correcta letra D.

Ahora veamos las pruebas personalizadas:



Grafica 2: Resultados entrenamiento personalizado.

En esta grafica vemos que el porcentaje de aciertos es más elevado, incluso en las letras críticas como lo eran antes la letra A y D. También observamos que obtenemos muy pocos falsos positivos, con lo cual nuestro sistema no dará demasiamos errores.

El problema con la letra D es el anteriormente mencionado en el apartado anterior.

7.4.2.1 Conclusiones:

Para nuestro juego hemos decidido usar los gestos que mejor más acierto nos dan, con lo cual hemos elegido los siguientes:

Avanzar

Figura 7.16: Juego Avanzar.

Retroceder

Figura 7.17: Juego Retroceder.

Izquierda

Figura 7.18: Juego Izquierda.

Derecha

Figura 7.19: Juego Derecha.

Rotar Izquierda

Rotar Derecha

Figura 7.20: Juego Rotar izquierda.

Figura 7.21: Juego Rotar derecha.

Disparar

Figura 7.22: Juego Disparar

7.4.3 Pruebas de sistema

La lógica del juego necesita recibir las acciones desde el reconocedor y llamar a los elementos correspondientes. En el sistema completo se realizaron las siguientes pruebas:

- Ordenador sin Leap Motion instalado: al acceder al juego desde un ordenador que no tiene instalado los *drivers* de Leap Motion las funcionalidades son nulas, ya que no podrá iniciar el escuchador.

- Ordenador sin Leap Motion conectado: al igual que en la prueba anterior se no se obtiene nada.

- Realización del juego empleando Leap Motion: en un ordenador con Leap Motion correctamente instalado. Se inicia correctamente y permite un funcionamiento completo durante la realización del juego.

7.5 Pruebas en entorno 3D:

Un plan de pruebas elaborado nos garantiza un mínimo de calidad para el proyecto. En esta sección enumeraremos las pruebas que se han realizado sobre la plataforma en su totalidad.

Diseñar casos de prueba para un videojuego como OgreTank es una tarea complicada ya que estamos simulando continuamente un mundo lleno de elementos que interactúan entre sí y hay métodos que se ejecutan muchas veces en un sólo segundo. Por supuesto, las pruebas son absolutamente necesarias, ya que nos ayudan a desarrollar software de mayor calidad.

La mayoría de módulos han podido ser probados de forma independiente en pruebas anteriores. Entre ellos se encuentran el reconocimiento de gestos, carga de datos y la grabación de estos.

- Pruebas de integración

  A medida que el desarrollo de los módulos finalizaba, se procedían a realizar pruebas de integración. Interesaba el sistema realizara la tarea de forma correcta, no solo entre los submodulos sino en conjunto del sistema.

  El mayor problema fue la rapidez del cálculo del DTW y el renderizado. Debido a que en nuestra aplicación de obtener gestos, el tiempo no es una variable critica, pero aquí sí.

Otro de los problemas encontrados es a la hora de cambiar de estado la variable de nuestro gesto, debido a que siempre debería estar asignada a una ID para que nuestro tanque estuviera siempre en movimiento.

Después de varias pruebas hemos obtenido que el retardo sea menos de 1 segundo aunque se necesita un tiempo de adaptación para su correcto manejo.

- Pruebas de jugabilidad

  Se pidió a colaboradores externos que probaran el videojuego. Tras estas sesiones se les preguntó por aspectos como la capacidad de respuesta del control, la comodidad de juego, dificultad y otros parámetros. Posteriormente, se analizaron las respuestas de todos los colaboradores y se llevó a cabo un proceso reestructuración para ajustar la calidad.

  La mayoría de colaboradores encontró el juego bastante complicado. Debido a que ese pequeño retardo hace que la adaptación no sea instantánea al juego.

8. CONCLUSIONES

Con este proyecto se persigue la investigación de un nuevo dispositivo como es Leap Motion, además de usar un entorno grafico como es OGRE3D. Para conseguir este objetivo, se ha realizado una profunda investigación de la API de Leap Motion y también de OGRE3D, ya que son dos tecnologías que no están en nuestros planes de estudio.

Incluimos también en este proyecto el uso de DTW (Dynamic Time Warping), algoritmo bastante interesante en el reconocimiento de gestos, tanto estáticos como dinámicos. Por ello hemos realizado pruebas con dos tipos de gestos, como hemos

mencionado en el apartado 7.4.1 y 7.4.2, sencillo y avanzados, para hacer que nuestro juego sea lo más intuitivo posible.

Los objetivos han sido alcanzados en su mayoría, se ha realizado un detector con un alto porcentaje de acierto y se ha complementado su uso en un entorno 3D donde conseguimos un manejo fluido mediante gestos.

Cabe destacar, que todo se ha programado en lenguaje C++, el cual tenía una base, pero he tenido que ampliar mis conocimientos sobre todo en procesamiento multihebra del que no tenía nociones. También se ha utilizado el entorno Visual Studio 2012, donde he adquirido bastante experiencia en la creación de proyectos, inclusión de librerías, paths etc.

Personalmente, estoy bastante satisfecho con el trabajo realizado, debido a que he ampliado mis conocimientos en programación, entornos gráficos y redacción de documentación.

9. LINEAS FUTURAS

En este tipo de proyectos donde se usa una tecnología nueva como es la de Leap Motion, existe un amplio abanico de aplicaciones y mejoras, si nos centramos en nuestro proyecto se pueden destacar:

9.1 Creación de una interfaz 3D.

Debido a los escasos conocimientos de creación de interfaces gráficas y el poco tiempo para realizarlos, se ha decidido realizar nuestro proyecto en consola de Windows, no obstante sería importante desarrollar una interfaz donde podamos visualizar los gestos (Figura 9.2) que estamos realizando en tiempo real para nuestro grabador/reconocedor de gestos e incluso para la iniciación a nuestro juego.

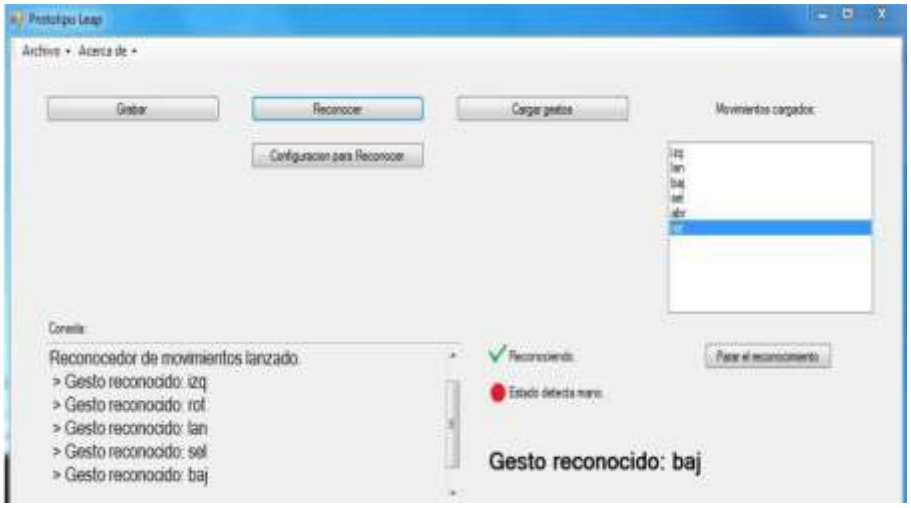


Figura 9.1: Interfaz reconocedor gestos.

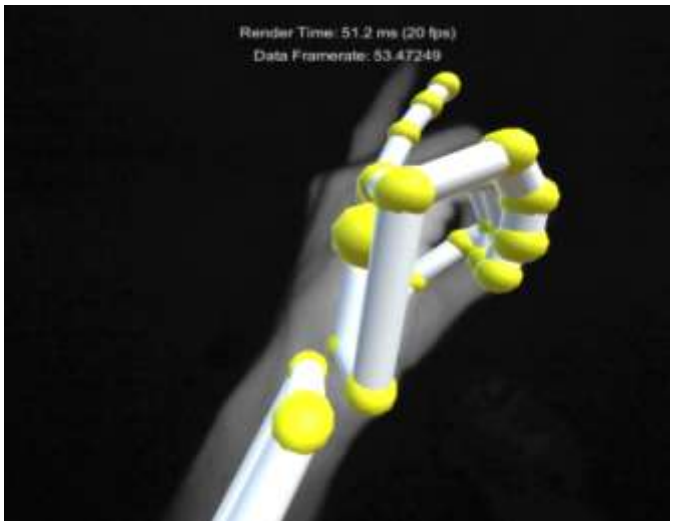


Figura 9.2: Visualizar en tiempo real gesto.

9.2 Utilización de otros métodos de clasificación.

En nuestro proyecto solo hemos utilizado el algoritmo DTW, pero podíamos incluir otros métodos de clasificación como son:

- **Modelos Ocultos de Markov.**
- **Redes Neuronales.**
- **Máquina de estados.**

Con estos métodos podríamos aumentar nuestro porcentaje de positivos y sería interesante incluirlos para hacer nuestro algoritmo más eficiente.

9.3 Uso de Leap Motion en otros dispositivos.

Este dispositivo como hemos realizado en nuestro proyecto se puede utilizar interfaz de manejo sobre juegos, debido a esto se puede aplicar en dispositivos como:

9.3.1 Portátil con Leap Motion integrado:

HP [19] ya ha apostado por la integración del dispositivo en sus ordenadores. (Ver figura 9.3 para más detalles). Uno de los problemas encontrados es su tamaño, por eso se ha rediseñado y como podemos ver en la imagen se han reducido sus dimensiones para su uso en ordenadores.

Figura 9.3: Ordenador con Leap Motin integrado.

9.3.2 Realidad Virtual:

Leap Motion, con el proyecto Orion [20], nos intenta introducir en la Realidad Virtual. Ya se han realizado pruebas junto a Oculus Ritft, dando buenos resultados. Estamos ante una nueva era de control gestual.

Figura 9.4: Oculus Rift con Leap Motion integrado.

10. ANEXOS

10.1 Instalación Leap Motion

Para empezar a utilizar este dispositivo tenemos que instalar el software específico. Para ello tendremos que dirigirnos a: https://www.leapmotion.com/setup?lang=es y nos aparecerá lo siguiente

Figura 10.1: Instalación Leap Motion.

Como podemos observar justo debajo podemos ver un botón para descargar el controlador para Windows (también está disponible para Mac y Linux), pulsamos sobre él y lo descargamos.

Una vez descargado, ejecutamos el asistente de instalación (Figura 10.2):

Figura 10.2: Ventana principal instalación Leap Motion.

Cuando haya terminado el proceso de instalación, para ver que todo ha salido correctamente, nos dirigimos a la parte inferior derecha de nuestro escritorio, y tendríamos que ver el siguiente icono

Figura 10.3: Icono Leap Motion.

Este icono puede tomar tres colores:

- El color verde significa que todo está correcto.
- El color naranja nos hace referencia a que la superficie del dispositivo está sucia.
- El color negro nos indica que no está el dispositivo conectado.

Clicando con el botón derecho sobre este icono podemos ver algunas opciones como son la configuración, visualizador, etc, indicadas en la figura 10.4.

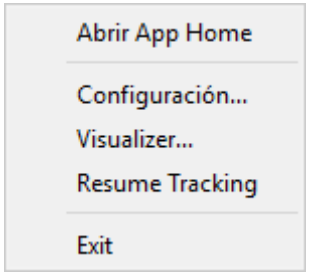


Figura 10.4: Opciones Leap Motion.

Podemos observar que nos ha creado un icono en nuestro escritorio, se trata de *Leap Motion App Store*, en donde podemos encontrar aplicaciones para nuestro dispositivo.

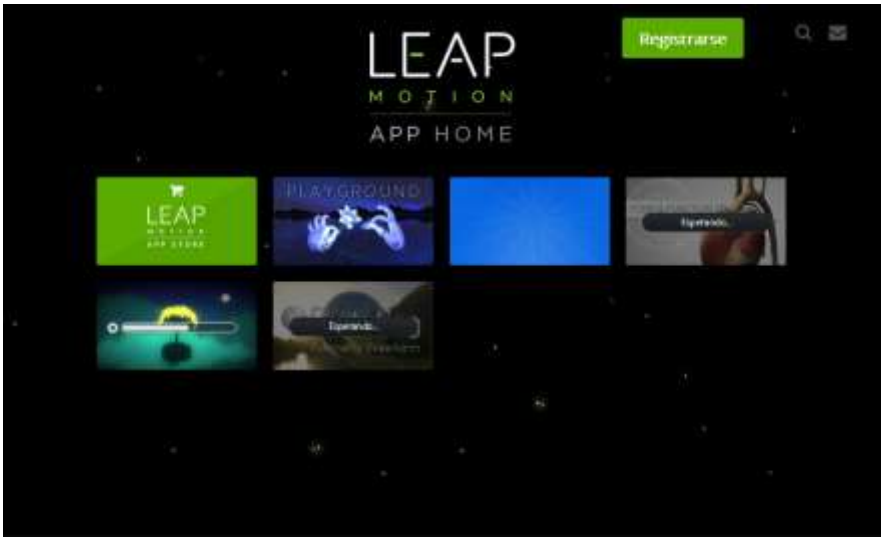


Figura 10.5: App Home Leap Motion

10.2 Instalación Ogre3D

En este apartado vamos a explicar cómo instalar el SDK OGRE pre compilado para Windows. (También está disponible para Mac y Linux). Nos dirigimos a http://www.ogre3d.org/download/sdk

Aquí podemos observar las siguientes versiones:

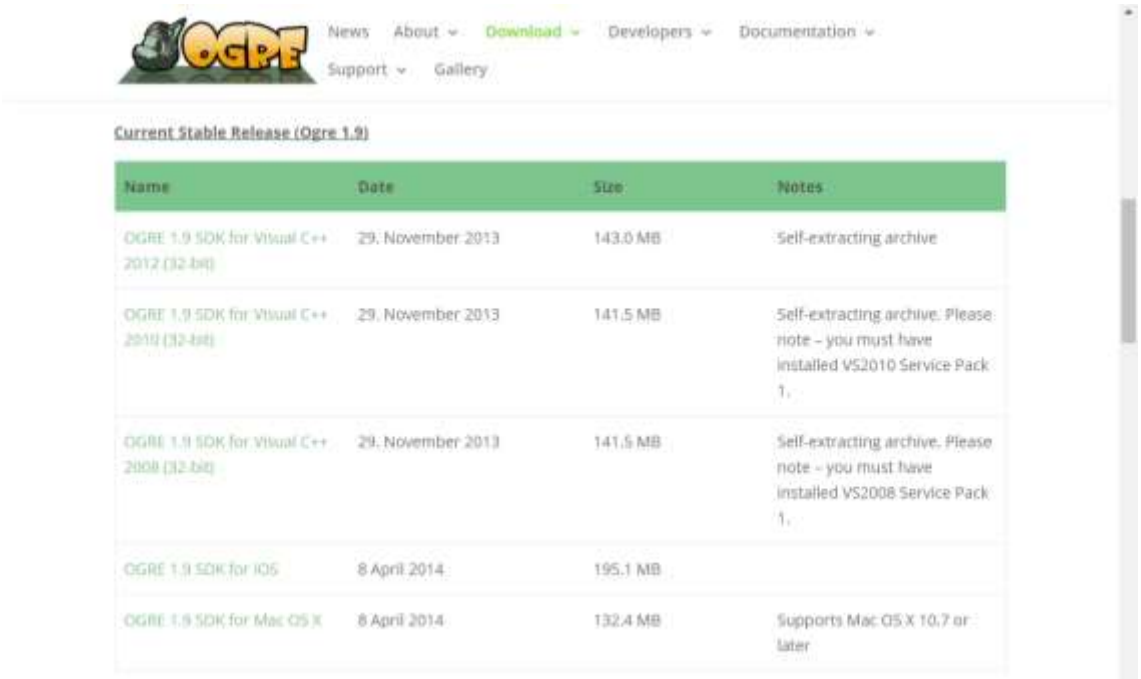


Figura 10.6: Versiones OGRE3D.

Dependerá de la versión de Visual C++ que tengamos instalada, en nuestro caso será 2012.

Una vez descargada, ejecutamos la aplicación y nos aparecerá lo siguiente:

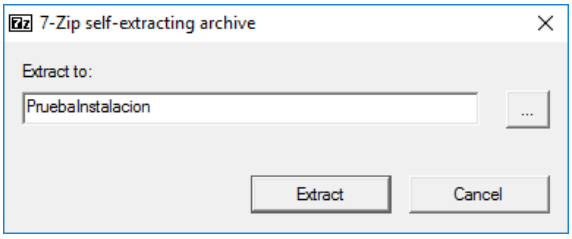


Figura 10.7: Extracción archivos OGRE3D.

Elegimos la ruta correcta y extraemos los archivos. Una vez completamos nos dirigimos hacia la carpeta y copiamos la ruta.

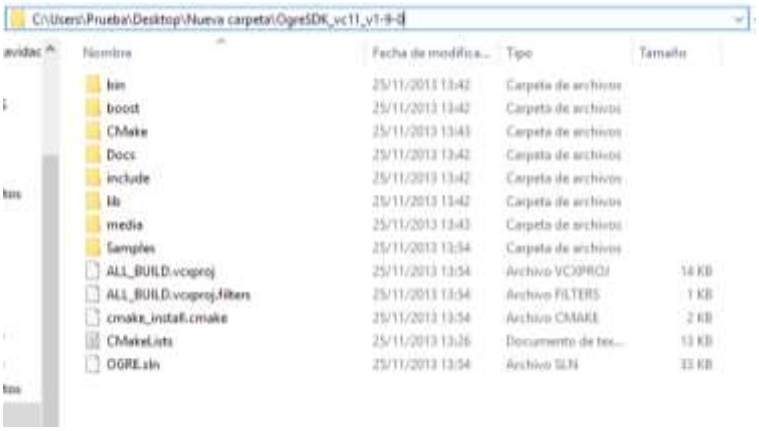


Figura 10.8: Carpeta descomprimida OGRE3D.

Para poder utilizar Ogre necesitamos establecer una variable de entorno OGRE_HOME. Esta variable nos permite cambiar fácilmente entre diferentes versiones y poder usarla diferentes usuarios. Para establecer esta variable nos dirigimos a *Panel de control\Sistema y seguridad\Sistema.* En la parte izquierda clicamos en *Configuración Avanzada del Sistema.*

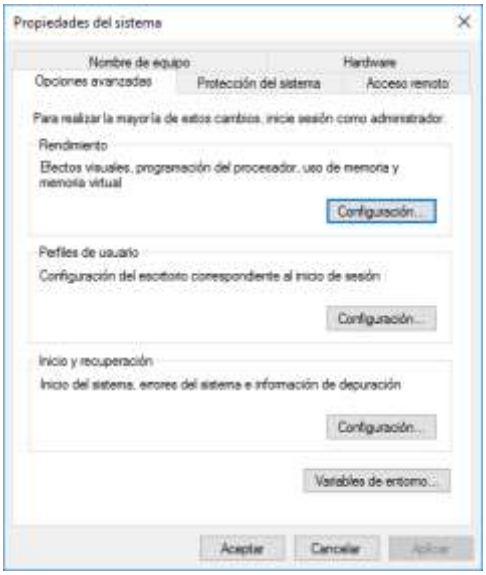


Figura 10.9: Configurar variable de entorno OGRE3D.

Dentro de variables de entorno creamos una *Nueva* y nos aparecerá:

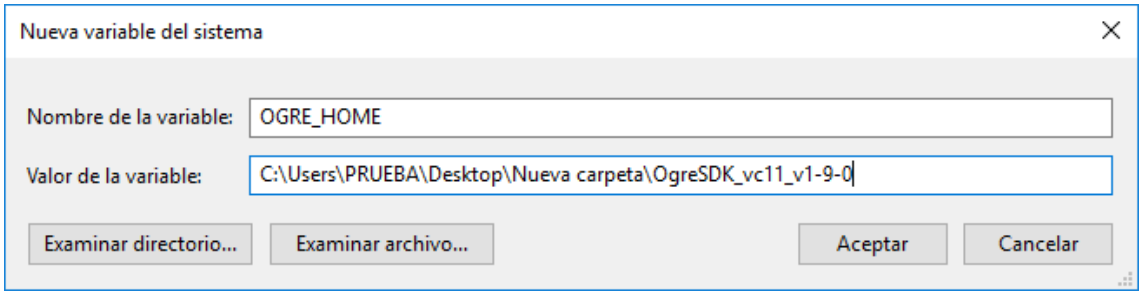


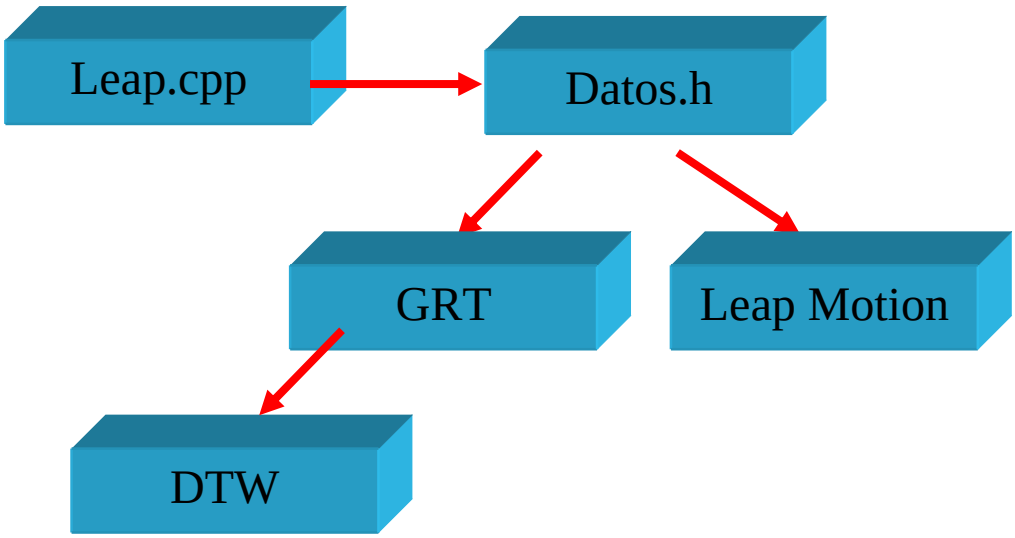
Figura 10.10: Crear una variable de entorno OGRE3D.

Aceptamos y después reiniciamos el ordenador para que los cambios sean efectivos. Ya tendremos listo Ogre3D para utilizarlo.

10.3 Manual de referencia

En este apartado vamos a explicar las funciones realizadas para utilizar nuestro reconocedor.

Aquí mostramos la jerarquía que seguiría nuestro reconocedor:



A continuación se muestran las funciones usadas en cada una:

1. **Leap.cpp**

   Se encarga de mostrar el menú principal

   - int main()

     Inicializamos variables globales y mostramos menú principal.

     Parámetros de entrada: ninguno.

     Parámetros de salida: ninguno.

2. **Datos.h**

   Se encarga de recoger todos los datos necesarios de Leap Motion, además de guardar, entrenar y cargar la base de datos de gestos.

   - int guardar()

     Inicializamos los variables necesarias y guardamos todos los datos recogidos por Leap Motion en el archivo DTWTrainingData.txt.

Parámetros de entrada: ninguno.

Parámetros de salida: ninguno.

- static void real()

  Habilita un escuchador y vamos guardando los datos recogidos en tiempo real y los comparamos con las plantillas anteriormente guardadas. Devolviendo por pantalla el gesto predicho.

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

- DTW cargar()

  Esta función se encarga de cargar el archivo anteriormente creado DTWTrainingData.txt y crear el modelo para comprar gestos denominado DTWModel.txt. Ademas copia el archivo resultante a la carpeta donde se encuentra el ejecutable de nuestro juego.

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

- DTW defecto()

  Esta función realiza la carga del modelo por defecto que hemos creado para la comparación de gestos. Nos sirve para no tener que entrenar el sistema.

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

- void AltEnter()

  Con esta función pretendemos que una vez que ejecutemos nuestro .exe se nos abra en pantalla completa.

  Parámetros de entrada: ninguno.

  Parámetros de salida: ninguno.

- virtual void onConnect(const Controller&)

  Nos sirve para indicarnos que el dispositivo Leap Motion esta conectado a nuestro ordenador.

  Parámetros de entrada: Controller.

  Parámetros de salida: ninguno.

- virtual void onFrame(const Controller&)

  Esta función es de las mas importantes, debido a que se encarga de inicializar las variables de las características que necesitamos de las manos y guardarlas en una serie de vectores.

  Parámetros de entrada: Controller.

  Parámetros de salida: ninguno.

Después de explicar el código desarrollado, también debemos introducir las librerías necesarias para el uso de GRT (Gesture Recognition Toolkit). Para ello tendremos que agregar la carpeta adjunta GRT a nuestro proyecto.
De ella vamos a explicar la función básica que usamos que son:

1. DTW.cpp

   - DTW(booluseScaling, useNullRejection, nullRejectionCoeff,UINT rejectionMode,bool dtwConstrain,double radius,bool offsetUsingFirstSample,bool useSmoothing,UINT smoothingFactor, double nullRejectionLikelihoodThreshold);

     Esta función se encarga de definir los parámetros de decisión que utilizaremos para nuestro algoritmo de decisión. Definimos parámetros como escalado, factor de suavizado, coeficiente de descarte etc.

Por último, para la integración con nuestro entorno 3D, hemos utilizado el proyecto indicado en [17]. Donde vamos a destacar la función que hemos modificado para la integración de nuestro reconocedor en el entorno 3D.

1. UserInput.cpp
    - void UserInput::Think(float time)

      Esta funcion se encarga de realizar los movimientos del tanque en el entorno 3D. Para ello por defecto tiene asignado teclas, sin embargo, nosotros hemos cambiado esos valores por gestos que nos ira devolviendo nuestro reconocedor.

10.4 Manual de usuario

Una vez explicadas las funciones que usa nuestro reconocedor de gestos en el apartado 6.6, vamos a enumerar las acciones necesarias para que nuestro juego funcione correctamente.

Recomendado: Entrenamiento personalizado y ejecución juego.

Para la realización de un entrenamiento se deberían seguir las siguientes opciones.

1. Entrenar Gestos.
    - Elegir qué tipo de gestos queremos entrenar.
    - Realizar los gestos mostrados en pantalla. Ver apartado *7.4.1* y *Conclusiones*
    - Realizar cada gesto 3 veces consecutivamente.
2. Entrenar Sistema.
3. Jugar Juego.

No recomendado: Entrenamiento por defecto y ejecución juego.

Esta sería una opción rápida para la ejecución del juego.

1. Cargar entrenamiento por defecto.
2. Jugar Juego.

Una vez entrenado el sistema, nos quedaría la parte más divertida, **JUGAR.**

OgreTank se basa en el control de tu propio tanque, con el objetivo final de eliminar a todos los demás tanques e ir subiendo de nivel. Para ello tendremos que ir disparando a todos ellos.

Se controla con los siguientes gestos, explicados en *Conclusiones*.

Avanzar

Figura 10.12: Juego Avanzar.

Retroceder

Figura 10.13: Juego Retroceder.

Izquierda

Figura 10.14: Juego Izquierda.

Derecha

Figura 10.15: Juego Derecha.

Rotar Izquierda

Figura 10.16: Juego Rotar izquierda.

Rotar Derecha

Figura 10.17: Juego Rotar derecha.

Disparar

Figura 10.18: Juego Disparar

11. PLIEGO DE CONDICIONES Y ESTUDIOECONÓMICO

Se denomina pliego de condiciones a un documento contractual, de carácter comprensivo y obligatorio donde se establecen las condiciones o cláusulas que se aceptan en un contrato de obras o servicios, una concesión administrativa, una subasta, etc.

En un pliego de condiciones se indica cómo y con qué hay que hacer realidad los proyectos de obras y servicios que se contratan. En el pliego que se concuerda y firma, contiene las relaciones que existirán y que tienen que cumplirse, entre el propietario y el ejecutor de cualquier proyecto, servicio o concesión administrativa. Este documento debe contener toda la información necesaria para que el proyecto llegue a buen fin de acuerdo con las características constructivas del mismo, indica las

condiciones generales del trabajo, la descripción y características de los materiales a utilizar, los planos constructivos de existir estos, y la localización de la obra o servicio. También señala los derechos, obligaciones y responsabilidades de las partes que los suscriben. Señala, así mismo, como se desarrollará el trabajo y como se resolverán los conflictos que puedan surgir. Normalmente los pliegos de condiciones se dividen en varias partes, siendo las más usuales las siguientes:

• Pliego de condiciones generales: esta parte del documento debe incluir la descripción general del contenido del proyecto, los criterios o aspectos normativos, legales y administrativos a considerar por las empresas que intervengan, listados de planos que componen el proyecto, etc. En España la normativa que se aplica es los pliegos de condiciones generales es la norma UNE 24042.

• Pliego de especificaciones técnicas: dispone de dos apartados perfectamente diferenciados: o Especificaciones de materiales y equipos: donde deben estar bien definidos todos los materiales, equipos, máquinas, instalaciones, etc. que se utilizarán en el proyecto. La definición se hará en función de códigos y reglamentos reconocidos. Las especificaciones hacen referencia a Normas y Reglamentos nacionales tipo (UNE, Normas MOPU, NBE, etc.) o internacionales (DIN, ISO, etc.). O Especificaciones de ejecución: es este apartado del pliego se hace constar como será realizado el proyecto, es decir, su proceso de fabricación o construcción a partir de los materiales que serán utilizados.

• Pliego de cláusulas administrativas: en este apartado del pliego se determina la forma de medir las partes ejecutadas del proyecto, valorarlas y pagarlas.

En este caso, se presentará un pliego de condiciones técnicas donde se realizará un breve resumen de las especificaciones materiales y equipos necesarios para la ejecución de este trabajo.

11.1 Pliego de condiciones técnicas

La parte principal de desarrollo del sistema, es el dispositivo Leap Motion, cuyas especificaciones mínimas para su uso son:

- Windows 7/8/10
- Mac OS X 10.6 Snow Leopard
- AMD Phenom o Intel Core i3
- 2 GB RAM
- USB 2.0
-

El software necesario para la ejecución de la aplicación, con las especificaciones impuestas por Leap Motion, no habría problema en ejecutarlo.

11.2 Estudio económico

Este estudio determina la cantidad de recursos económicos que se necesitan para que el proyecto sea realizado. El presupuesto se dividirá en partes identificadas por el tema que engloban y el resultado total.

A continuación se detalla el importe al que asciende el presupuesto:

11.2.1 Costes materiales:

Se denominan costes materiales a aquellas herramientas físicas usadas para implementar la labor de trabajo. A continuación, se expone un presupuesto:

UD.	CONCEPTO	P.UNITARIO	SUBTOTAL
1	Ordenador portatil Procesador i5 + Velocidad procesador 2GHz Tarjeta gráfica NVIDIA GeForce 6200 Disco duro 256 GB 4GB RAM	499 €	499€
1	Leap Motion	89.99 €	89.99€
		Importe Total:	588.99 €

11.2.2 Costes técnicos:

En este apartado se especifica el coste técnico del software utilizado para el funcionamiento de la aplicación. Seguidamente, se expone el presupuesto estimado.

UD.	CONCEPTO	P.UNITARIO	SUBTOTAL
1	Sistema Operativo W10	30 €	30€
1	Microsoft Visual C++ 2012 Express Edition	0€	0€
1	OgreSDK vc11 v1.9	0€	0€
1	LeapMotion SDK 3.13	0€	0€
		Importe Total:	30 €

11.2.3 Honorarios:

Podemos estimar que la duración del desarrollo y creación de la aplicación pueden suponer unos 3 meses, empleando solo los días laborables, por tanto 20 días al mes y trabajando unas 8 horas al día.

El costo de hora de trabajo, calculando seguridad social y otros gastos indirectos, estimamos un costo de 30 €.

Por tanto:

60 días laborables x 8 horas diarias x 30 € = 14.400 €

11.2.4 Presupuesto final:

UD.	CONCEPTO	P.UNITARIO	SUBTOTAL
1	Costes Materiales Costes Tecnicos	30 €	30€
1	Costes Hardware y Software	588.88€	588.99€
1	Honorarios	14.400€	14.400€
		Importe Total:	15.018,99 €

12. BIBLIOGRAFIA

[1] API Reference Developer Documentation. © 2016, Leap Motion, Inc. Disponible en: https://developer.leapmotion.com/documentation/cpp/api/Leap_Classes.html

[2] Sistema de reconocimiento gestual para una unidad de medición inercial universidad politécnica de valencia. Año 2010. 107 pág. Gustavo Vidal Moreno. https://riunet.upv.es/bitstream/handle/10251/8609/MEMORIA_PFC_GUSTAVO_VIDAL.pdf

[3] How Does the Leap Motion Controller Works? © 2016 Leap Motion, Inc. Disponible en: http://blog.leapmotion.com/

[4] Técnicas de visión estereoscópica para determinar la estructura tridimensional de la escena. Universidad Complutense de Madrid. Martín Montalvo Martínez. Año 2010. 97 pág. Disponible en: http://eprints.ucm.es/11350/1/T%C3%A9cnicas_de_visi%C3%B3n_estereosc%C3%B3pica_para_determinar_la_estructura_tridimensional_de_al_escena.pdf

[5] Start Learning Ogre. Tutorials. Disponible en: http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Tutorials&structure=Tutorials

[6] How Does the Leap Motion Controller Works? © 2016 Leap Motion, Inc. [en línea] [última consulta: Marzo 2016]. Disponible en: http://blog.leapmotion.com/

[7] Introducción a Ogre3D. Disponible en: http://tirwal.blogspot.com.es/2012/06/ogre3d-parte-i-introduccion-ogre3d.html

[8] ¿Qué es Ogre3D?. Disponible en: http://isaaclacoba.github.io/tinman/posts/introduccion-ogre3d/introduccion-a-ogre3d.html

[9] Start Learning Ogre. Tutorials. Disponible en: http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Tutorials&structure=Tutorials

[10] N. Gillian, R. Benjamin, S. O`Modhrain. Recognition Of Multivariate Temporal Musical Gestures Using N-Dimensional Dynamic Time Warping. 6 Pag. Disponible en: http://www.nickgillian.com/papers/Gillian_NDDTW.pdf

[11] Openframeworks DTW Example. Disponible en: http://www.nickgillian.com/wiki/pmwiki.php/GRT/OpenframeworksDTWExample

[12] MAKIKO FUNASAKA, YU ISHIKAWA, MASAMI TAKATA, AND KAZUKI JOE. Sign Language Recognition using Leap Motion Controller. Año 2016. 7 pag. Disponible en: http://worldcomp-proceedings.com/proc/p2015/PDP7080.pdf

[13] MICHAEL NOWICKI, OLGIERD PILAREZYK y otros. Bachelor's thesis: Gesture recognition library for leap motion controller [en línea]. Poznan, 2014 Disponible en: http://www.cs.put.poznan.pl/wjaskowski/pub/theses/LeapGesture_BScThesis.pdf

[14] JestPlay [en línea]. 20 Diciembre 2013 Disponible en: https://github.com/jaanga/gestification/tree/gh-pages/cookbook/jest-play

[15] Visualising hand using Leap Motion and Ogre3D. 18 de Julio 2014. Disponible en: https://github.com/anilbey/HandVisualiser

[16] Simply static gesture recognition for #3DJam. 22 de Noviembre 2015. Disponible en: https://github.com/Gesturio/Gesturio.github.io

[17] Tank Game for Game Engineering 3D course. 6 de Junio 2014. Disponible en: https://github.com/WofloW/OgreTank_plus

[18] Vocabulario básico lenguaje de signos lengua española. Disponible en: http://ssaacg2.blogspot.com.es/p/vocabulario-basico.html

[19] HP integra el sistema de control gestual Leap Motion. Disponible en: http://www.xataka.com/ordenadores/hp-integra-el-sistema-de-control-gestual-leap-motion-en-su-portatil-hp-envy-17

[20] Proyecto Orion Leap Motion. Disponible en: https://www.unocero.com/2016/02/22/orion-la-nueva-apuesta-de-leap-motion-para-realidad-virtual/