

Aplicaciones Multimedia para Rehabilitación Cognitiva

Grado en Ingeniería Informática



Trabajo Fin de Grado

Autor:

Javier Ortega Bastida

Tutor/es:

José García Rodríguez

Julio 2016



**Agradecimientos**

*Dedico un especial agradecimiento a mi tutor y a todos los miembros del equipo que me han ayudado en este proyecto.*

**Índice de contenido**

[1. Motivación y Contexto 9](#_Toc454537684)

[2. Introducción 11](#_Toc454537685)

[2.1. Estructura de la Memoria 13](#_Toc454537686)

[3. Estado del Arte 14](#_Toc454537687)

[3.1. Marco Teórico 14](#_Toc454537688)

[3.1.1. Fundamentos Teóricos 14](#_Toc454537689)

[3.1.2. Diferentes Tipos de Algoritmos 15](#_Toc454537690)

[3.1.3. Proyectos Teóricos 16](#_Toc454537691)

[3.2. Proyectos Técnicos 17](#_Toc454537692)

[3.2.1. Leap Trainer.js 18](#_Toc454537693)

[3.2.2. Leap en el Automóvil 18](#_Toc454537694)

[3.2.3. Rock Paper Scissors Island! 19](#_Toc454537695)

[3.2.4. Blocks 20](#_Toc454537696)

[4. Objetivos 21](#_Toc454537697)

[5. Metodología 22](#_Toc454537698)

[5.1. Dispositivos 22](#_Toc454537699)

[5.1.1. Leap Motion 24](#_Toc454537700)

[5.2. Base de Datos 25](#_Toc454537701)

[5.3. Dynamic Time Warping 27](#_Toc454537702)

[5.4. Transformación de Coordenadas 29](#_Toc454537703)

[6. Desarrollo 30](#_Toc454537704)

[6.1. Funcionamiento SIRVAMED 30](#_Toc454537705)

[6.2. Implementación del Entrenamiento 32](#_Toc454537706)

[6.2.1. Transformación de Coordenadas 35](#_Toc454537707)

[6.3. Reconocimiento de gestos 39](#_Toc454537708)

[6.3.1. Dynamic Time Warping 42](#_Toc454537709)

[6.4. Aplicación de Rehabilitación: Sopa de Letras 44](#_Toc454537710)

[6.5. Análisis de Resultados 47](#_Toc454537711)

[7. Conclusiones 51](#_Toc454537712)

[8. Bibliografía 52](#_Toc454537713)

**Índice de Figuras**

Figura 1.1. Representación de un paciente que sufre daño cerebral adquirido 9

Figura 2.1. Aspecto del sistema SIRVAMED 11

Figura 2.2. Interfaz de Usuario 12

Figura 3.1. Flujo de datos en el reconocimiento de gestos 15

Figura 3.2. Representación de una mano en un modelo 3D 15

Figura 3.3. Representación de un modelo basado en el esqueleto 16

Figura 3.4. Representación de una mano en un modelo basado en la apariencia 16

Figura 3.5. Aplicación Leap Trainer.js 18

Figura 3.6. Ejemplo de inmtegración del Leap en vehículos 19

Figura 3.7. Captura RPS Island 19

Figura 3.8. Captura de Blocks 20

Figura 5.1. Dispositivo Kinect 22

Figura 5.2. Dispositivo Eye Tracker 23

Figura 5.3 Dispositivo Leap Motion 23

Figura 5.4. Ejemplo Leap Motion 24

Figura 5.5. Hardware Leap Motion 24

Figura 5.6. Pantalla de Configuración de los dispositivos 26

Figura 5.7. Dynamic Time Warping 27

Figura 5.8. Pseudocódigo del Algoritmo Dynamic Time Warping 28

Figura 5.9. Como detecta Leap Motion las coordenadas 29

Figura 6.1. Creación de páginas con QtCreator 30

Figura 6.2. Enlazar botones con métodos mediante señales 31

Figura 6.3. Ventana de Entrenamiento…………………………………………………………………. 32

Figura 6.4. Directorio de un Usuario…………………………………………………………….………. 33

Figura 6.5. Visualizador 3D Leap Motion……………………………………………………………… 34

Figura 6.6. Registrar Usuario 37

Figura 6.7. Identificar Usuario 38

Figura 6.8. Menú Principal 39

Figura 6.9. DTW entre joints 43

Figura 6.10. Ficha de Estimulación Cognitivia 45

Figura 6.11. Ficha de Estimulación Cognitivia Respuesta 45

Figura 6.12. Sopa de letras 46

Figura 6.13. Sopa de letras 2 46

Figura 6.14. Precisión del Leap Motion…………………………………………….…………………….49

**Índice de Tablas**

Tabla 6.1. Resultados de Precisión 47

Tabla 6.2. Resultado con gestos grabados por otro usuario 48

**Índice de Gráficas**

Gráfica 6.1. Resultados de Precisión 48

Gráfica 6.2. Resultado con gestos grabados por otro usuario 50

# Motivación y Contexto

Desde hace varios años, se ha observado un gran avance en el desarrollo de lo que podría denominarse las primeras “aplicaciones de nueva generación”, las cuales hacen uso de diferentes dispositivos con sensores como la cámara RGB-D denominada Kinect o los mandos de la consola Nintendo Wii que detectan el movimiento e incluso poses y gestos que realizamos. Lo que ha permitido, entre otras cosas, nuevas experiencias en el entretenimiento y rehabilitación de personas con discapacidades motoras o cognitivas. Conforme estas tecnologías se han ido asentando, se han utilizado en diferentes proyectos para alcanzar un mayor avance en el desarrollo de tecnologías de interacción Hombre-Computador HCI (*Human–Computer Interaction*), siendo el campo médico y más concretamente el de la rehabilitación uno de sus objetivos principales.

Existen diferentes daños que la rehabilitación desea paliar, siendo el daño cerebral uno de los más difíciles de reparar. Diferentes estudios han demostrado que los juegos, no sólo pueden funcionar como una ayuda para el aprendizaje en las escuelas, sino que también pueden utilizarse en las terapias para quienes padecen esta discapacidad. Al igual que un paciente con discapacidad motora inferior se le hace caminar como método de rehabilitación para “despertar” sus piernas y reactivarlas, los juegos ejercitan el cerebro y reparan conexiones haciéndolo trabajar.



Figura 1.1. Representación de un paciente que sufre daño cerebral adquirido

Se estima que en España hay cerca de cien mil personas que padecen daño cerebral [9], por lo que el objetivo de este Proyecto de Fin de Grado(*PFG*) es: desarrollar un sistema integrado de diferentes sensores que permitan una buena interacción hombre-máquina (en inglés *HCI*)*,* incluyendo juegos interactivos que sirvan de ayuda para la rehabilitación de estos pacientes. Pensar en desarrollar un sistema de soporte en su recuperación basado en la realización de aplicaciones que hagan uso de sensores de movimiento fue una motivación para realizar este trabajo y sumamente gratificante personalmente.

El valor de este Proyecto de Fin de Grado, no es solamente desarrollar un trabajo que presentar al final del curso, sino que además es participar en un proyecto nacional colaborando con un equipo de investigación del Instituto Universitario de Investigación en Informática (*IUII*) de la Universidad de Alicantecon un objetivo *real*, un sistema que los hospitales podrán utilizar para la rehabilitación de pacientes. Ser parte de un equipo, donde se integrará todo el trabajo que realice y estar en comunicación y colaboración con el resto de miembros.

# Introducción

El sistema desarrollado, recibe el nombre de SIRMAVED y consiste en un sistema multisensor (Figura 2.1) realizado por el *IUII* de la Universidad de Alicante con la colaboración con el Hospital Virgen del Rocío de Sevilla y que forma parte de la convocatoria de proyectos de retos de la sociedad del Ministerio de Economía y Competitividad.

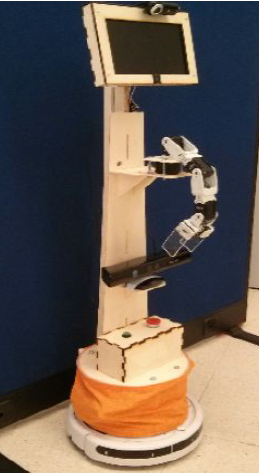


Figura 2.1. Aspecto del Sistema SIRVAMED

El objetivo de dicho sistema es ofrecer a los pacientes un entorno de rehabilitación mediante aplicaciones de juegos personalizado para que los usuarios lo puedan configurar según sus necesidades, ya que, la capacidad de comunicación difiere según el paciente. Por lo que incorporan múltiples dispositivos con sensores de movimiento.

El paciente o cuidador podrá elegir cuál es el o los dispositivos que prefiera utilizar para usar las aplicaciones y calibrar a su gusto. Se registrará en el sistema y su configuración quedará guardada para cargarla en usos posteriores del sistema.

Mediante el sistema SIRVAMED se ofrece un entorno, no sólo personalizado, también accesible para que el que el paciente pueda utilizarlo por sí mismo sin depender de nadie. Permite un reconocimiento de voz y rostro (Figura 2.2) para poder iniciar sesión sin ayuda, permitiendo de este modo que el usuario sea autónomo y los hospitales ahorren en recursos. Todo el sistema ha sido programado en *C ++* y se ha utilizado la librería *Qt* para la interfaz gráfica de las diferentes pestañas del sistema.



Figura 2.2. Interfaz de Usuario

Dicho sistema incorpora como sensores: una cámara *Kinect*, un sensor ocular *Eye Tracker* y un sensor de movimiento *Leap Motion* para que el usuario decida a qué juego jugar y con qué dispositivo. El dispositivo en el que se centrará esta memoria será el *Leap Motion* que emplearemos para la detección de gestos.

    El sistema, permitirá a los programadores desarrollar múltiples juegos de rehabilitación cognitiva y jugarlos independientemente del dispositivo que se utilice. No es necesario programar varias veces la aplicación por cada dispositivo. Los dispositivos emiten señales según la acción que han detectado, si han detectado la acción “aceptar”, emitirán la señal aceptar y el juego actuará en consecuencia a la señal emitida (independientemente del dispositivo que la haya emitido). El sistema tiene cuatro movimientos: aceptar, cancelar, siguiente y anterior.

## Estructura de la Memoria

A lo largo de la memoria seguiremos una estructura muy concreta de modo que el lector pueda realizar un correcto seguimiento de todos los aspectos que se han considerado en este proyecto final. Una vez explicada la motivación y justificación y presentada la introducción de este trabajo. Comentaremos el estado del arte de distintos proyectos que nos han servido de inspiración y guía para saber los aspectos a considerar al utilizar el *Leap Motion*. Acto seguido, plantearemos unos objetivos mínimos a conseguir. Seguiremos con la metodología, en la que haremos una breve descripción de los dispositivos que forman el sistema que sirve de entorno para la rehabilitación de pacientes, así como las diferentes técnicas que necesitaremos para el reconocimiento de gestos que realizaremos con dichos dispositivos. Luego mostraremos en el apartado de desarrollo todo el trabajo realizado y la forma de alcanzar los objetivos planteados, así como un análisis de los resultados obtenidos. Y, tras el apartado del desarrollo, llegaremos a unas conclusiones para corroborar si llegamos a dichos objetivos.

# Estado del Arte

En esta sección vamos a exponer una serie de trabajos realizados por otros desarrolladores dentro de un marco teórico y técnico. Han servido de inspiración tanto de los pasos a seguir como de los diferentes aspectos a tener en cuenta en la detección de gestos en general y del *Leap Motion* en particular. Más adelante, en la metodología expondremos el dispositivo y todos los que componen el sistema *SIRVAMED*.

## Marco Teórico

Vamos a contemplar los aspectos teóricos de diferentes estudios en la detección de gestos.

### Fundamentos Teóricos

La detección de gestos es un ámbito de estudio recurrente en la ciencia de la computación, haciendo uso de algoritmos matemáticos para ser capaces de detectar e interpretar dichos gestos. Enfocado, mayoritariamente, en el reconocimiento de expresiones faciales y gestos con las manos. Interpretando el lenguaje de signos [4], mediante cámaras y sensores, las máquinas consiguen ser capaces de entender e interpretar los gestos realizados por las personas, logrando de ese modo un mayor *HCI.*

Mediante la detección de gestos, algunos desarrolladores pretenden sustituir (o convivir) con el uso del teclado y del ratón. En nuestro caso, para pacientes con daño cerebral, les es imposible hacer uso de dichos hardware, por lo que ser capaces de interpretar sus acciones en las aplicaciones supone un remedio para dichas personas.

Observamos que el flujo de datos en la detección y el reconocimiento de gestos (Figura 3.1), se inicia en el hardware extrayendo las características representativas de los gestos generando entradas que son interpretadas en el middleware. Cuando el software reconoce el gesto genera una salida donde, si el gesto es reconocido, enviará la acción correspondiente y el hardware actuará en consecuencia al gesto detectado.

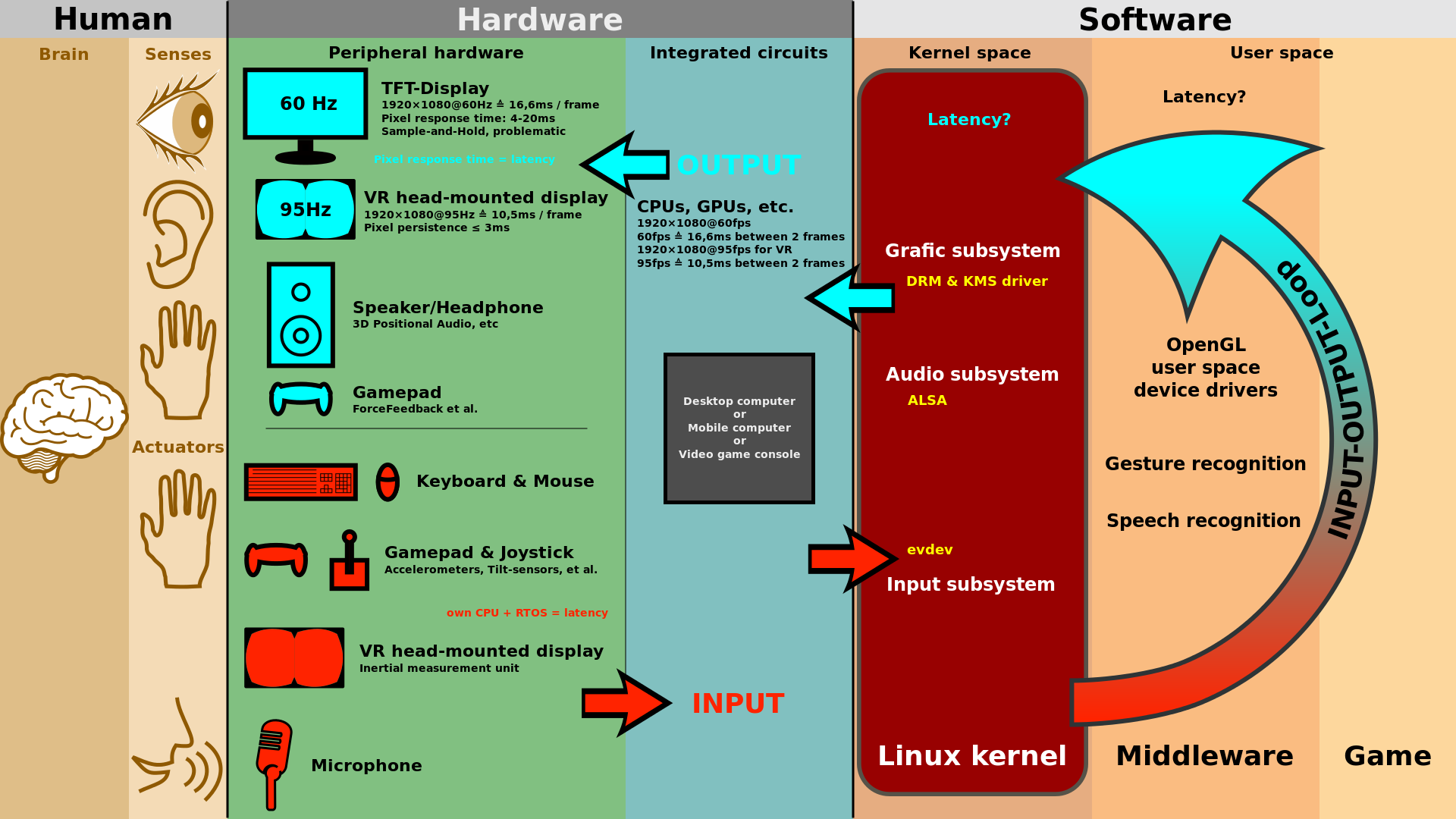


Figura 3.1. Flujo de datos en el reconocimiento de Gestos [5]

### Diferentes Tipos de Algoritmos

Podemos encontrar diversos modelos algorítmicos en la interpretación de un gesto. Pueden ser algoritmos basados en modelos 3D, donde se representa la mano que realiza el gesto como un conjunto de vértices (figura 3.2). Calculando la posición relativa de la malla generada se reconoce el gesto. Esta técnica requiere de altas prestaciones computacionales.

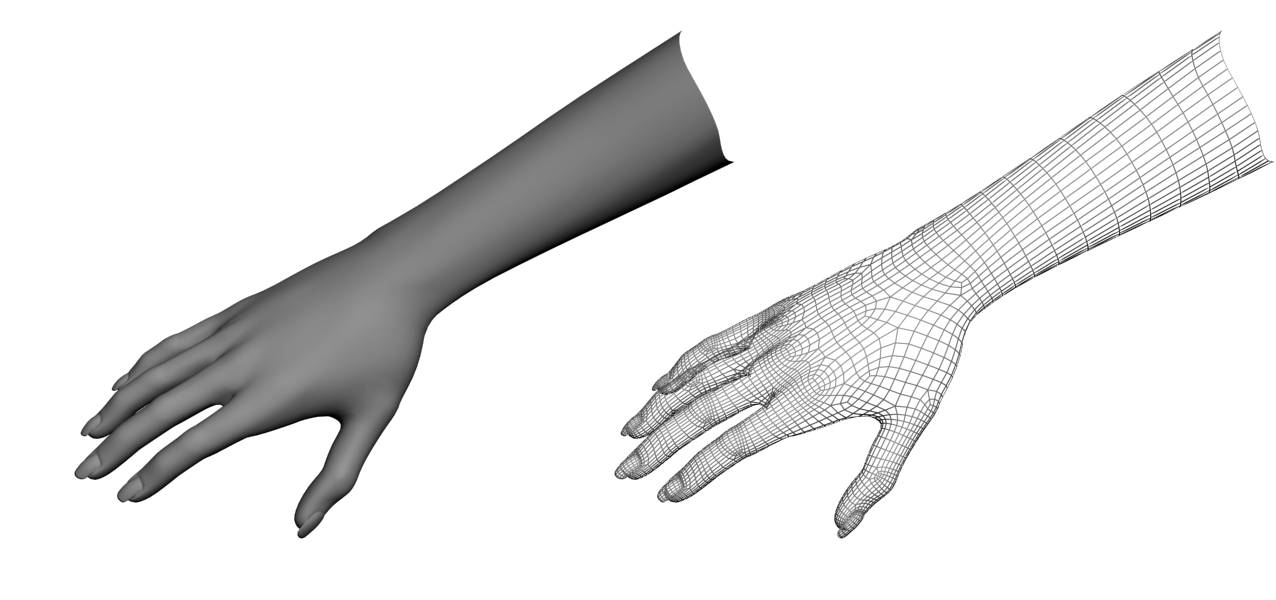


Figura 3.2. Representación de una mano en un modelo 3D [5]

También existen algoritmos basados en el esqueleto que reconocen los gestos mediante una representación del esqueleto virtual del cuerpo (Figura 3.3). A continuación, se realiza un análisis de los distintos segmentos del esqueleto, como el ángulo de las articulaciones, las posiciones o la orientación y la relación entre los segmentos. Estos algoritmos son más rápidos que los basados en modelos 3D.

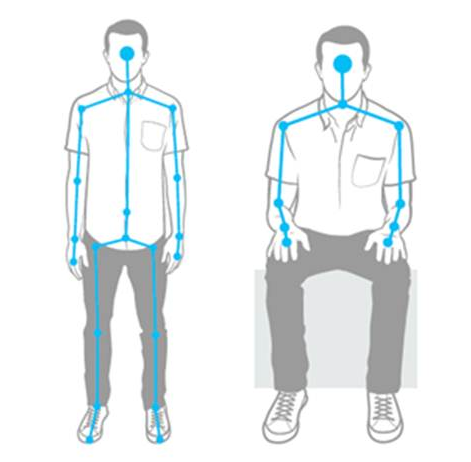


Figura 3.3. Representación de un modelo basado en el esqueleto [11]

Y los algoritmos basados en apariencia, que no hacen uso de una representación virtual de la mano (como las mallas de los modelos 3D o un esqueleto virtual en los basados en esqueletos). Lo que realizan es un procesado de las imágenes capturadas por las cámaras. Se hace un seguimiento del contorno de la mano y se compara con varias plantillas 2D (Figura 3.4).

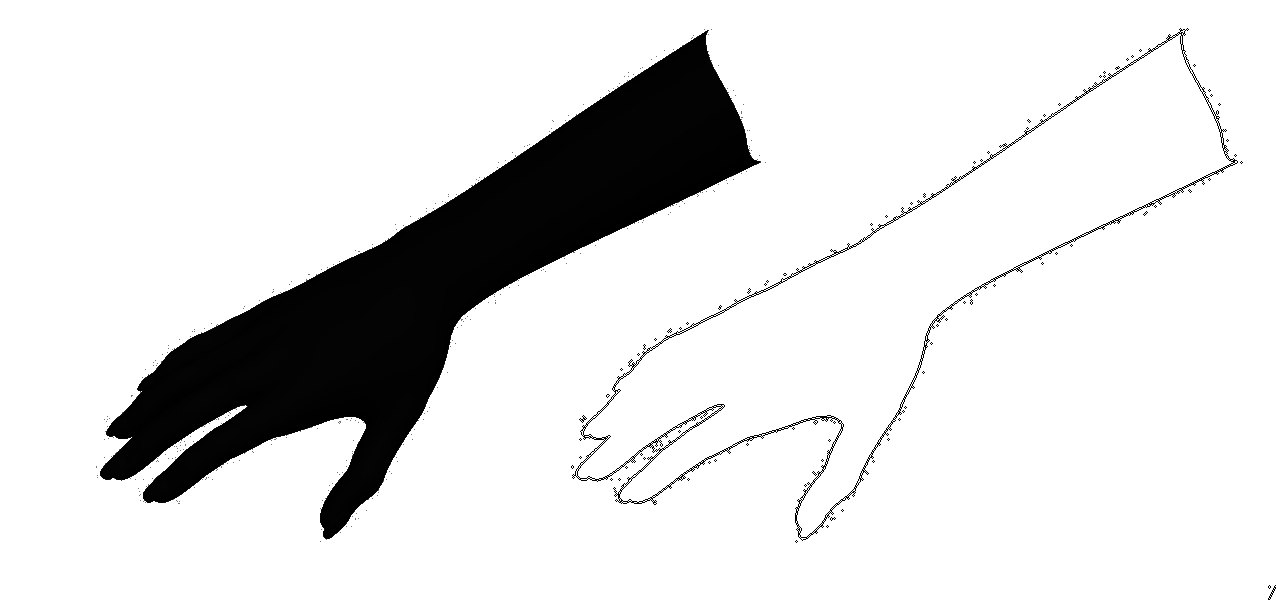


Figura 3.4. Representación de una mano en un modelo basado en apariencia [5]

### Proyectos Teóricos

Existe una gran variedad de artículos de investigación en los que se trata el reconocimiento de gestos. Vamos a presentar el par de artículos que más han influenciado en este *PFG*:

***“Recognizing Schaeffer's gestures for robot interaction”*** [7]

El artículo nos presenta un nuevo sistema de comunicación y detección del lenguaje de gestos de Schaeffer. Cuenta que hace uso de una cámara RGBD para capturar la información del cuerpo y reconocer los gestos. Este proyecto presenta, para pacientes con discapacidad cognitiva, que les cuesta hablar y necesitan de estos gestos para comunicarse, varios tonos o alarmas al detectar ciertos gestos. Detecta lo que los usuarios quieren expresar mediante la cámara de una Kinect v2.0, y los procesa un módulo de preselección de gestos. Del mismo modo que explicamos en la metodología, este sistema hace uso de *DTW* para comparar los gestos y devuelve el vecino más cercano (*Nearest Neighbor*). Para la disminución de resolución hace uso de K-Medias con 20 centroides. En definitiva, este artículo resume bastante bien el proceso que seguiremos en nuestro proyecto. Detección, procesamiento mediante *DTW* y devolver el vecino más cercano (el gesto que más se parece).

***“Gesture recognition using RFID technology” [8]***

El artículo expone un estudio donde propone la técnica *RFID (Radio Frequency IDentification),* que consiste en almacenar e identificar datos a través de etiquetas que ejercen como identificadores de dichos datos. Cuenta el artículo, que hace uso “de etiquetas pasivas, baratas y discretas que se pueden conectar fácilmente a la ropa del usuario”, que serán leídas por las antenas *RFID*. A través de estaslecturas, puede reconocer qué gestos está realizando el usuario. El autor nos cuenta que probablemente su trabajo es el primero en hacer un patrón de seguimiento de movimientos realizado con las manos mediante antenas *RFID.* A pesar de ciertas imprecisiones expone resultados donde podemos observar resultados de hasta el 93% de precisión.

## Proyectos Técnicos

A continuación, se presentan otros proyectos relacionados con el aquí expuesto que sirvieron de inspiración y orientación para desarrollarlo. Se tratan de proyectos similares que aprovechan el Leap Motion para el reconocimiento de gestos.

### Leap Trainer.js

Una aplicación muy interesante y con ciertas semejanzas a lo que se desea implementar es Leap Trainer (Figura 3.5). Programado en Java Script consiste en una interfaz donde el usuario, haciendo uso del Leap Motion, puede definir sus propios gestos [6].

****

Figura 3.5. Aplicación Leap Trainer.js

El cliente asignará un gesto, por ejemplo, el gesto “izquierda”, elige el nombre y a continuación realiza el gesto. Guardará los datos referentes a ese gesto (coordenadas, etc.…) y posteriormente, cuando realice el gesto guardado, lo reconocerá con la etiqueta que ha utilizado.

### Leap en el Automóvil

Una de las aplicaciones más interesantes encontradas, es la integración de la tecnología del Leap Motion en dispositivos para automóviles (Figura 3.6). Realizando gestos en el aire, es capaz de detectarlos para realizar múltiples opciones con el automóvil. Con estas cómodas acciones se puede encender la música o el aire acondicionado, cambiar el volumen reconocerá los gestos “presionar una acción” o “pasar a la siguiente”.



Figura 3.6. Ejemplo de Integración del Leap en vehículos [1]

### Rock Paper Scissors Island!

Hay una infinita cantidad de videojuegos que utilizan el *Leap Motion* (y más que habrá en un futuro). La gran mayoría consiste en representar las manos tal cual la detectan y las representan en el mundo 3D. Pero el juego RPS Island (Figura 3.7) hace un análisis para detectar los gestos, estos son “piedra”, “papel” o “tijeras”. Es cierto que esta aplicación se centra más en el aspecto visual, ya que los tres gestos que detecta son muy fáciles de lograr con el *Leap Motion*.



Figura 3.7. Captura de RPS Island [1]

### Blocks

En la página oficial de Leap Motion aparece una gran cantidad de aplicaciones que podríamos destacar. Por ejemplo, la aplicación de Blocks (Figura 3.8).

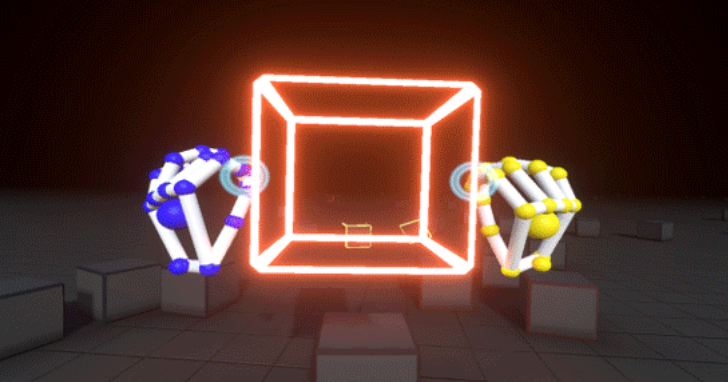


Figura 3.8. Captura de Blocks [1]

Para probar este juego, es necesario utilizar las últimas gafas de realidad virtual creadas por la empresa desarrolladora del *Leap Motion*. Éstas gafas hacen uso de la misma tecnología que utiliza el *Leap*. También reconoce ciertos gestos para realizar acciones. Para crear bloques, hay realizar el gesto de unir las manos y separarlas para lograr la creación, el dispositivo lee nuestros movimientos y reconoce el gesto. También hay que reconocer que con dichas gafas de realidad virtual pretenden solventar el problema de la precisión del dispositivo tan criticada por varios desarrolladores.

# Objetivos

Una vez establecida la necesidad de un proyecto para pacientes con daño cerebral y el estado del arte y todo el estudio previo en la detección de gestos que seguiremos en su desarrollo, debemos tener claros unos objetivos mínimos a alcanzar con este trabajo, tanto personales como técnicos antes de pasar a la metodología que seguiremos.

Los objetivos técnicos:

* El sistema deberá permitir al usuario poder utilizar la configuración que desee tanto la de por defecto como la personalizada.
* Crear los gestos personalizados para cada paciente. Guardarlos en su configuración y cargarlos al iniciar el sistema.
* Ser capaz de detectar con un mínimo de error y en el menor tiempo posible el gesto que realiza el usuario en tiempo de ejecución.
* Desarrollar un pequeño juego que puede ser probado independientemente del dispositivo que se esté utilizando.

Respecto a los objetivos personales, los que podríamos destacar serían:

* Aprender a trabajar en un verdadero equipo de desarrollo que sirva más adelante para la vida profesional. Colaborando estrechamente con el resto de miembros e integrando nuestras partes del proyecto.
* Aumentar los conocimientos de distintas tecnologías como el Leap Motion, la librería Boost, Qt u OpenCV.
* Pero la más importante y, como apuntamos en la introducción, la más gratificante, desarrollar un Proyecto de Fin de Grado que tenga como objetivo principal servir de ayuda en la recuperación de pacientes con daño cerebral. En pocas palabras, un proyecto que sea útil a otras personas.

# Metodología

A continuación, vamos a describir las herramientas, tecnologías y técnicas que se utilizarán para el desarrollo de este proyecto de fin de grado. Será necesario considerar varios aspectos para tener un sistema estable.

## Dispositivos

SIRVAMED ofrece a los usuarios diferentes dispositivos para navegar por su interfaz y ejecutar las aplicaciones que incorpora. Según la discapacidad del paciente, hará uso del dispositivo que satisfaga mejor sus necesidades. Los diferentes dispositivos han sido calibrados para que los usuarios puedan configurarlos libremente en la detección de sus acciones.

Incorpora una *Kinect 2.0* (Figura 5.1) para reconocimiento de gestos dinámicos y amplios de las manos, detectar rostros y hacer un reconocimiento de voz. Habrá pacientes que no podrán mover sus manos para las acciones. Mediante el reconocimiento de voz podrán indicar al sistema qué acción desean realizar [3].



Figura 5.1. Dispositivo Kinect

Un sensor ocular *Eye Tracker* (Figura 5.2) para rastrear el movimiento de los ojos. El paciente configurará el dispositivo para que este reconozca las acciones que realiza (por ejemplo, guiñar dos veces sobre un botón es equivalente a la acción “aceptar”).



Figura 5.2. Dispositivo Eye Tracker

Y un sensor de movimiento *Leap Motion* (Figura 5.3) para detectar las manos y los gestos que realicen a una distancia corta. Según qué gestos detecte realizará varias acciones [2].

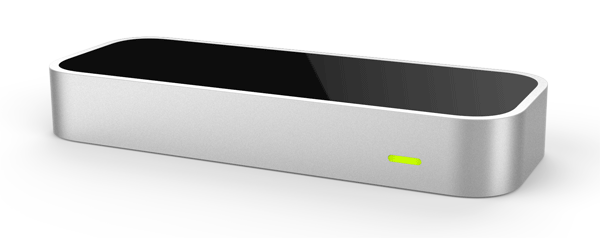


Figura 5.3. Dispositivo Leap Motion

Si bien todos los miembros del equipo de desarrollo hemos tenido que conocer cómo funcionan los diferentes dispositivos hardware que conforman SIRVAMED y sus correspondientes APIs de programación, cada uno se ha especializado en un dispositivo. Siendo el Leap Motion en el que me especialicé.

  Recordemos que la finalidad del sistema es poder programar aplicaciones para que el usuario las juegue y le sirvan en su recuperación utilizando el dispositivo que considere oportuno. Sin embargo, para llegar a ese punto de “centrarse sólo en la programación del juego” en el caso del Leap Motion, es necesario tenerlo controlado, programando algoritmos para el reconocimiento de gestos no predefinidos por él como explicaremos a continuación.

Por lo que este TFG se centrará en el dispositivo que me especialicé: el Leap Motion. Más adelante, vislumbraremos todos los factores a considerar con este dispositivo.

### Leap Motion

****

Figura 5.4. Ejemplo Leap Motion

Leap Motion [1] es un pequeño dispositivo (75 mm de largo y 25mm de ancho) (Figura 5.4) el cual captura el movimiento generado por nuestras manos en un entorno 3D.

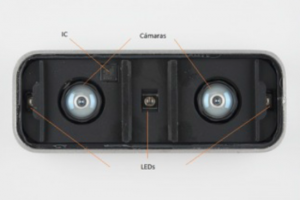


Figura 5.5. Hardware Leap Motion

Está compuesta por dos cámaras, tres LEDs y un microcontrolador (Figura 5.5) que otorgan un largo radio de alcance para detectar nuestros movimientos. Leap Motion está constantemente generando *frames* con la información que captura: número de manos, de dedos extendidos, dirección a la que se apunta, coordenadas x,y,z, si sostiene algún artefacto y lo más importante en nuestro caso, si ha realizado un gesto y si éste forma parte de los definidos por Leap Motion. La web oficial ofrece toda la documentación necesaria de la API para programar, así como el SDK [1].

    Cuando un usuario se registra, todos los dispositivos tienen una configuración por defecto. Queremos que el sistema sólo acepte tres movimientos: aceptar, cancelar y siguiente. Por defecto, el Leap Motion toma estos movimientos como: mover la mano hacia abajo para aceptar, hacia arriba para cancelar y realizar un círculo para cambiar el foco al siguiente botón. Sin embargo, con usuarios con daño cerebral esta configuración por defecto no va a funcionar siempre, pues habrá pacientes que no podrán realizar estos gestos.  Así pues, se les debe permitir que ellos configuren sus propios gestos para aceptar, cancelar y siguiente. Estos gestos, lo más probable es que Leap Motion los clasifique como gesto desconocido, por lo que no podremos procesarlos directamente con el dispositivo.

Deberemos utilizar la información de los eventos que son lanzados continuamente por el controlador (manos, dedos, coordenados, rotación, dirección, etc.…) para guardar dicha información en un archivo XML haciendo uso de la serialización que ofrece la librería Boost que hemos incorporado al proyecto. Cuando el usuario inicie sesión cargará la información de los gestos guardados. Pero el problema no se queda ahí, pues tratándose de pacientes con problemas cognitivos raramente realizarán dos veces exactamente el mismo gesto, será necesario un algoritmo de reconocimiento de gesto para hacer una comparación entre el gesto detectado con el guardado en la configuración del usuario.

## Base de Datos

Para tener una buena fuente de datos de muestra, se le pide al usuario que realice varias veces el gesto a la hora de guardarlo. Tres veces por gesto, por lo que el usuario tendrá un vector de gestos (se representa mediante una clase propia), concretamente doce. Ya que, si sólo guardásemos una muestra por gesto, tendríamos poca información para comprobar, y al tener tres muestras, permitimos variaciones en los movimientos.

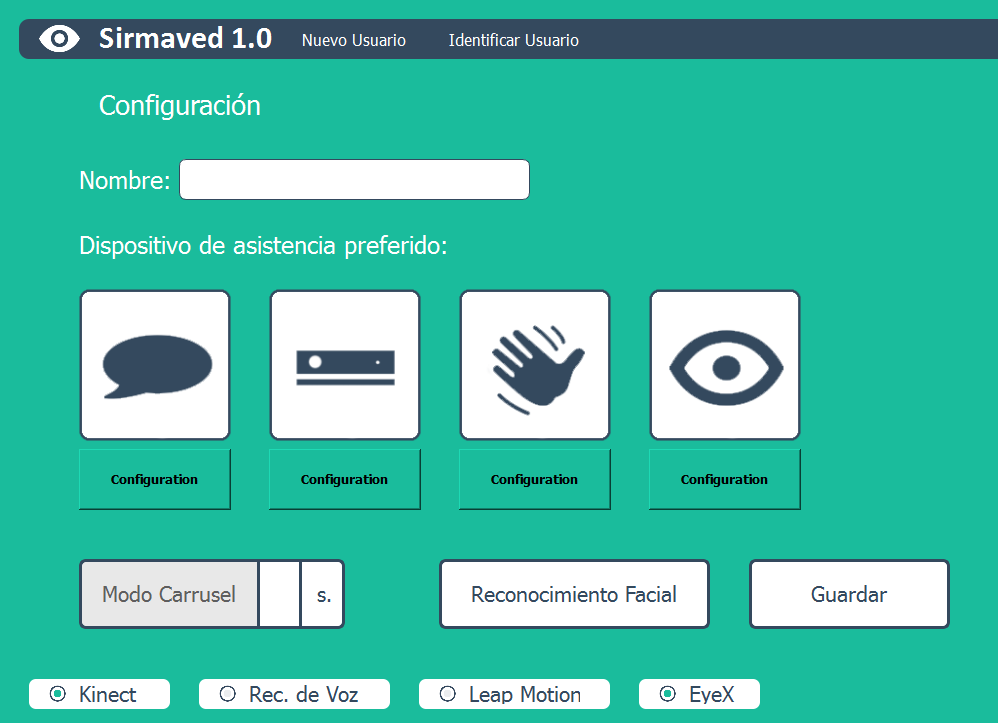


Figura 5.6. Pantalla de configuración de los dispositivos

Más adelante, cuando el usuario inicie sesión y se carguen sus datos (Figura 5.6), así como sus gestos (si no desea utilizar la configuración por defecto), podrá decidir a qué juego jugar.

  Una vez inicie el juego (o mientras esté navegando por las opciones de SIRVAMED), el sistema deberá detectar en tiempo de ejecución qué gesto es el que ha realizado para emitir la señal correspondiente. Para reconocer los gestos y compararlos con los de la base de datos del usuario se utiliza el algoritmo Dynamic Time Warping (*DTW)*.

## Dynamic Time Warping

Dynamic Time Warping es necesario, pues es posible que el usuario realice el mismo gesto a velocidades distintas, usando DTW normalizamos y reducimos ese efecto. Se rellena una matriz en la que los ejes son las dos series de gestos a comparar y las celdas se rellenan con la distancia entre los valores de las filas y columnas sumando el mínimo de los valores adyacentes anteriores. Devolvemos la última posición de la matriz, siendo esta la distancia (diferencia entre el gesto detectado y con el que queremos comparar). Dos trayectorias gestuales similares, aunque una desplazada en el tiempo con respecto a la otra tendrá una distancia muy baja (Figura 5.7).

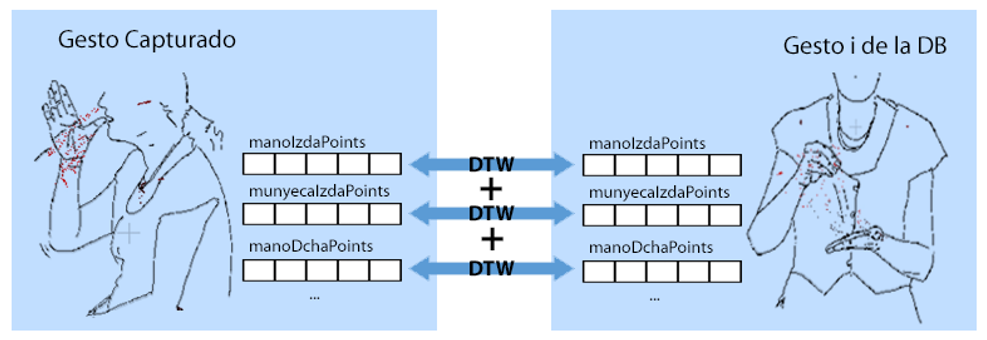


Figura 5.7. Dynamic Time Warping

Podemos diferenciar entre gestos estáticos (un único *frame*) y gestos dinámicos (una sucesión de *frames*). Cuando los usuarios guardan sus gestos, estos son equivalentes a 60 *frames*. Para captar los que realiza en tiempo de ejecución habrá que usar un temporizador que cada x segundos lea los movimientos del usuario y cree el gesto dentro de ese margen. Cuando el sistema ha creado el gesto en tiempo de ejecución, lo comparará con los gestos incorporados en el vector de gestos del usuario utilizando DTW. Un gesto dinámico, como hemos indicado, lo representamos como una sucesión de *frames*, donde cada *frame*, guardamos el número de manos, la posición x,y,z de la palma y de la muñeca,  la posición x,y,z de los dedos y de sus 4 falanges y la posición x,y,z del brazo. Se guardan todos estos datos, pero realmente con las coordenadas de la palma y de los dedos es suficiente a la hora de comparar gestos. Cuando se ha creado el gesto capturado en tiempo de ejecución, deberemos compararlo con todos los gestos del usuario. Para comprobar los gestos, deberemos comparar la información de cada frame que conforma el gesto con todos los frames del gesto guardado. Deberemos hacer una comparación de los diferentes joints (usamos seis joints, las posiciones de los dedos y de la palma de la mano), eso significará 6 matrices (una por joint a comparar) y, si tenemos doce gestos guardados, serán en total 72 matrices, ello puede conllevar cierto tiempo de ejecución. Respecto al método de clasificación usaremos K Vecinos. Realizaremos pruebas con diferentes K a devolver. Calculamos la distancia del gesto que el usuario está realizando con todos los del modelo y devolvemos la clase de aquel (o aquellos) que se parece más.

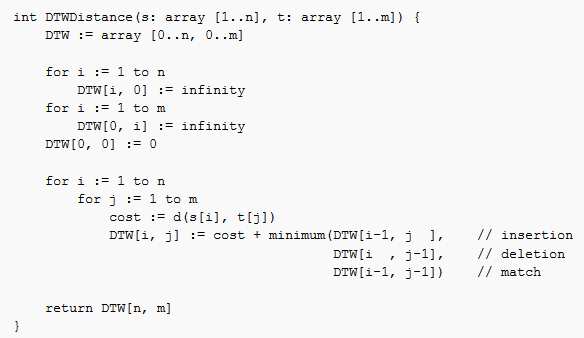


Figura 5.8. Pseudocódigo del Algoritmo Dynamic Time Warping

El algoritmo DTW, como puede apreciarse en el pseudocódigo (Figura 5.8), es de complejidad cuadrática, por lo que debemos encontrar modos de acelerarlo. Por ejemplo, fijar un umbral a partir del cual asumimos que la distancia es demasiado alta y los gestos no se parecen lo suficiente, ya que es posible que el gesto con la distancia más corta no tenga nada que ver con el gesto realizado. También, en el momento de comparar el gesto con los del modelo, guardar la distancia del gesto más parecido que llevamos de momento, para que, si al comparar con otro, y a la mitad éste tiene una distancia mayor deje de comparar. Se trata de ir poniendo condiciones restrictivas que reduzcan el tiempo de comparación.

## Transformación de Coordenadas

Otra cosa a tener en cuenta a la hora de detectar gestos es que debemos definir nuestro propio sistema de referencia para asegurarnos que el gesto sea detectado correctamente independientemente de en qué parte del espacio se realice (Figura 5.9).

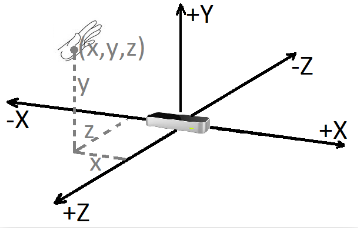


Figura 5.9. Cómo detecta Leap Motion las coordenadas

Las coordenadas que obtenemos de las distintas articulaciones siempre son con respecto al dispositivo. Por lo que el mismo gesto más alejado o más cerca del Leap, devolverá unas coordenadas diferentes, eso, nos dará una distancia considerable al aplicar DTW, por lo que tomaremos la palma de la mano como el origen y a partir de ahí calcular las coordenadas de los dedos. Por lo que se utilizará una matriz de transformación de coordenadas.

Una vez el Leap Motion ya está preparado para ser utilizado por los pacientes, el siguiente paso es programar juegos de estimulación cognitiva que sirvan para su rehabilitación.

Estos son los pasos que conforman la metodología que seguiremos en el desarrollo de este Proyecto de Fin de Grado.

# Desarrollo

En esta sección vamos a detallar el desarrollo del proyecto: la implementación realizada, los apartados técnicos existentes, detalles de los algoritmos utilizados en la detección de gestos y el flujo del programa. También mostraremos resultados de precisión que evaluaremos para poder visualizar la eficacia del algoritmo y la robustez del sistema.

## Funcionamiento SIRVAMED

Para trabajar con el sistema SIRVAMED, éste ofrece una serie de distintos hilos de ejecución correspondientes a cada dispositivo que lo forma: el *Eye Tracker*, la *Kinect* y el *Leap Motion* y con la funcionalidad de la librería *Qt* podemos diseñar las diferentes ventanas por las que navegaremos (como las páginas de configuración de perfiles o los juegos de estimulación cognitiva) y añadir los botones que emitirán distintos eventos. Todas las páginas que añadamos tendrán algunos elementos en común, como el botón de regresar atrás.

Podemos ver un ejemplo de creación de páginas de *Qt* haciendo uso del *IDE* de *QtCreator* (Figura 6.1) con sus diferentes elementos: *push button, text box*, etc…

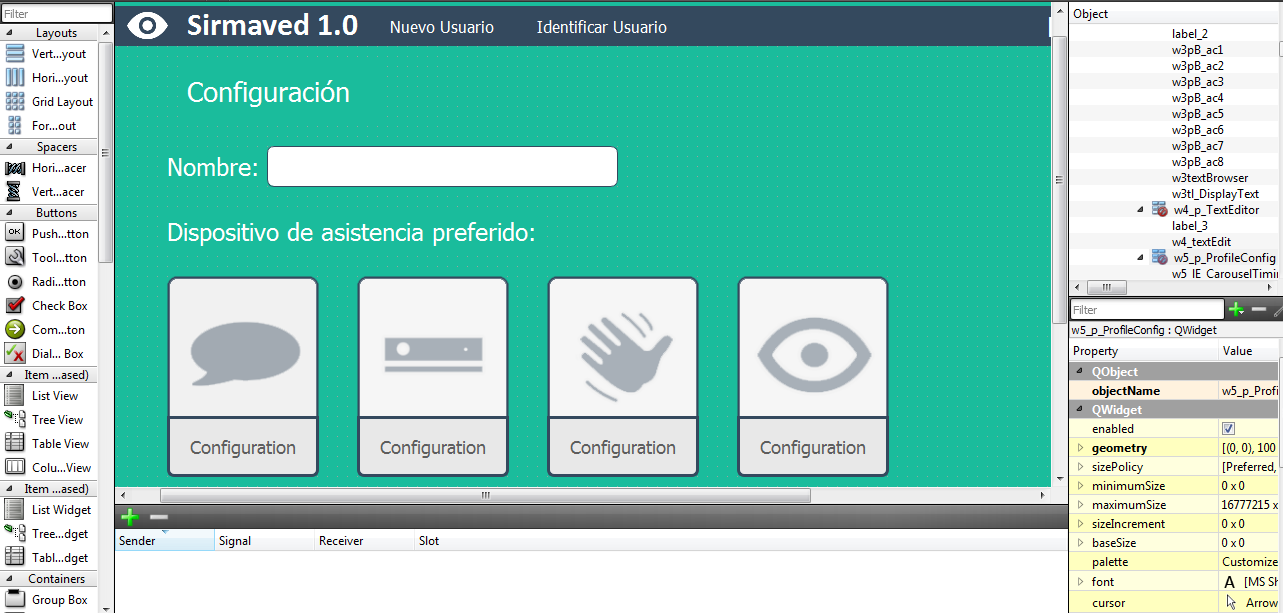


Figura 6.1. Creación de páginas con QtCreator

Todas las páginas (con sus elementos) se guardarán un fichero *.ui* que hace uso de un contenedor de pestañas de *Qt*, *“stackedWidget”,* donde contendrá toda la información del diseño de las ventanas. En nuestro código podremos acceder a los elementos del *.ui* y trabajar con ellos.

Cada hilo lanzará las señales o eventos que serán procesados por la clase principal para actuar en consecuencia. En dicha clase, “conectaremos” los botones de las páginas a métodos que serán llamados cuando se efectúe sobre ellos la señal que les hayamos asociado (Figura 6.2). Según el botón que clickeamos con el ratón o si emitimos la señal “Siguiente” o “Previo” moveremos el foco por los distintos botones de la ventana. Y sobre el botón donde se encuentre el foco, si emitimos la señal de “Aceptar” este lanzará el evento “*clicked()*” sobre el botón donde tengamos el foco posicionado.

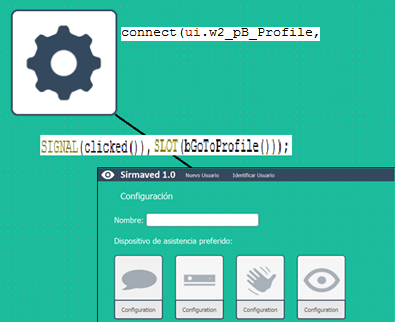


Figura 6.2. Enlazar botones con métodos mediante señales

Nuestro dispositivo (*Leap Motion)* lanzará la señal oportuna según el gesto que detecte: Siguiente, Previo, Aceptar y Cancelar.

## Implementación del Entrenamiento

Si se decide navegar por el sistema empleando el *Leap Motion,* como hemos ido comentando a lo largo de la memoria, el usuario tiene dos opciones: usar los gestos por defecto: aceptar, bajar la mano de forma; siguiente, dibujar un circulo con el dedo índice en sentido horario; previo, un circulo en sentido anti horario. O configurar sus propios gestos para que sean reconocidos en tiempo de ejecución. Cuando se registra el paciente, aparece la siguiente ventana con las opciones de configuración de cada dispositivo que vimos anteriormente (Figura 5.6). De todas formas, aunque el usuario no configure el dispositivo cuando se registra, puede cambiarla más adelante si lo desea.

Al seleccionar la configuración del *Leap Motion*, nos vamos a la pestaña de entrenamiento (Figura 6.3):

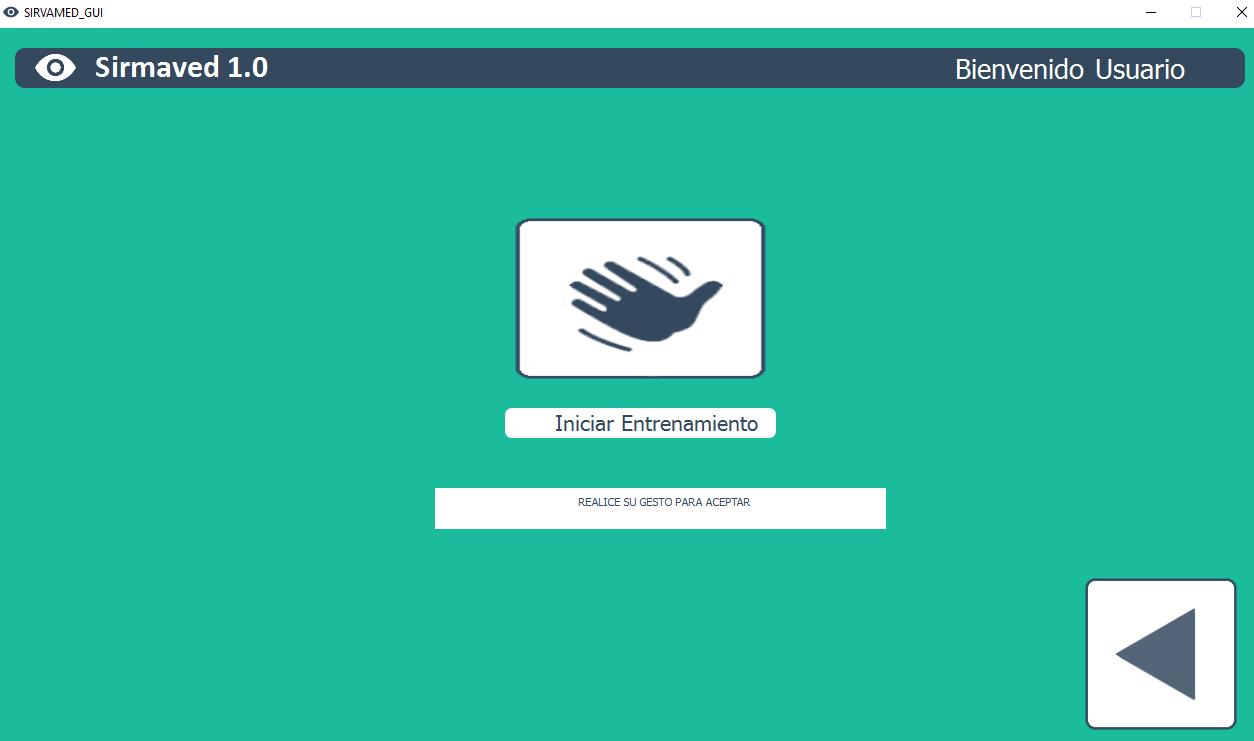


Figura 6.3. Ventana de Entrenamiento

Al iniciar el entrenamiento, el sistema le pedirá al usuario que realice tres veces cada gesto. El primero será aceptar: el usuario realizará tres veces aceptar (habrá un pequeño intervalo de tiempo entre cada gesto), seguido de siguiente, anterior y por último cancelar (debido a que el gesto cancelar no era necesario en la versión final este gesto se descartó). El gesto no estará completado hasta que haya almacenado exactamente 60 *frames*, una vez almacenados, guardará la muestra, después pasará a la siguiente. Si el usuario retira la mano antes de llegar a los 60, el sistema no sigue almacenando *frames* hasta que el dispositivo vuelva a detectar una mano.

Realizamos hasta tres veces cada gesto para más adelante, al reconocerlos cuando se navegue por la aplicación, tengamos una muestra lo suficientemente representativa de cada gesto. Recordemos que los pacientes con discapacidad difícilmente realizarán el mismo gesto exactamente igual siempre, por lo que es necesario disponer de más de una muestra por gesto a la hora de comparar, para poder abarcar posibles variaciones cuando se efectúe la acción.

Como comentamos en la metodología que emplearemos, los gestos se guardarán en la carpeta personal del usuario. Cada usuario registrado en el sistema dispondrá de un directorio (identificado por el id del paciente, asignado automáticamente al registrarse) donde almacenará la información de su configuración personalizada por cada dispositivo.

Cada gesto se almacenará en un archivo *xml* con la información de la mano a lo largo de los 60 *frames* guardados. Podemos ver el ejemplo de un directorio (Figura 6.4) con los gestos guardados tras el entrenamiento (también podemos apreciar que el propietario de este directorio ha realizado también una configuración personalizada con la Kinect (archivo “gestures”).

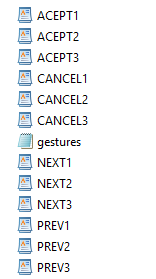


Figura 6.4. Directorio de un usuario

Vamos a conocer exactamente qué guardamos del entrenamiento de gestos, podemos apreciar en la imagen (Figura 6.5) cómo representa *Leap Motion* la información de la mano que capta en el visualizador 3D.

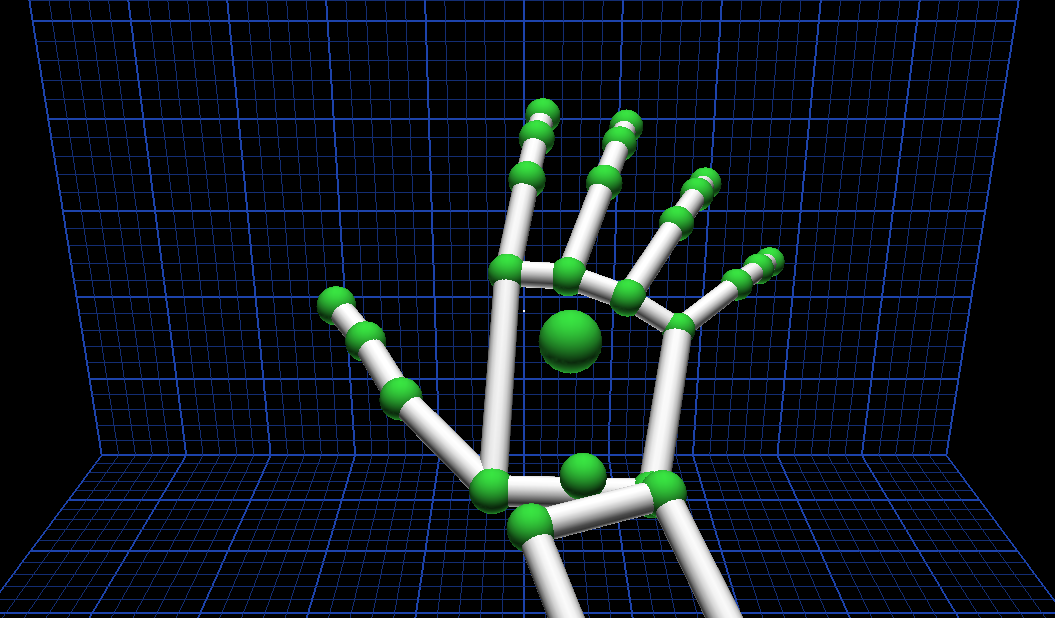


Figura 6.5. Visualizador 3D Leap Motion

La comparación de gestos se basa, en esencia, en comparar coordenadas. De ese modo sabemos si se ha seguido una misma trayectoria desde las coordenadas de la posición inicial a la final. *Leap Motion* representa la mano a través de múltiples *joints*, nosotros optamos por conocer las coordenadas x, y, z de la palma de la mano y de los cinco dedos, seis *joints* en total. Así pues, cada *xml* tendrá una cantidad de sesenta ítems (60 *frames*) donde cada uno guardará las coordenadas de los seis *joints*.

Pero no es suficiente, si queremos un sistema que compare los gestos de forma robusta es necesario realizar una *Transformación de Coordenadas* sobre las posiciones de los *joints* de esos 60 *frames.* Si no realizamos esta transformación, deberemos realizar el gesto en la misma posición con respecto al *Leap Motion* en la que realizamos el entrenamiento. No queremos que las posiciones de los *joints* sean con respecto al dispositivo como origen de coordenadas. Nuestro gesto, aunque lo realicemos más cerca o lejos del *Leap*, debe detectarlo de igual forma.

### Transformación de Coordenadas

Para realizar la transformación de coordenadas, debemos establecer un nuevo sistema de referencia, ello implica una serie de pasos que expondremos a continuación.

El primer paso es establecer los nuevos ejes x, y, z del nuevo sistema relativo a la palma de la mano. Definimos tres puntos mediante las ecuaciones (1) para más adelante obtener dos vectores cuya normal representará el nuevo eje z. Estos tres puntos serán: Va: el dedo pulgar, Vb: la palma de la mano y Vc: el dedo de en medio. Elegimos estos tres ya que forman 90 grados y el nuevo eje z parece perpendicular a la palma. Tomamos las coordenadas de estos tres *joints* del primer *frame*:

A continuación, ecuación (2), como hemos dicho, calculamos los dos vectores (Vbc: vector entre la palma y el dedo de en medio y Vba: vector entre la palma y el pulgar).

Y obtenemos el nuevo eje z a partir del producto vectorial de los dos vectores, ecuación (3):

𝑍𝑒 = 𝑉𝑏𝑐 × 𝑉𝑏𝑎 (3)

Calculamos los nuevos ejes x e y a partir del nuevo eje z, ecuación (4):

(4)

Una vez establecido el nuevo sistema de referencia, pasamos a obtener la matriz de transformación. Para ello calculamos los ángulos entre los ejes de la cámara de profundidad del *Leap* y de nuestro recién creado sistema de referencia.

Ecuación (5), Xc, Yc y Zc representan los ejes de la cámara.

(5)

Calculamos el ángulo entre cada eje (el eje x de la cámara con el eje x de nuestro nuevo sistema de referencia. La y con la y, etc…), ecuación (6):

𝛼𝑥 = 𝜃(𝑋𝑒, 𝑋𝑐) 𝛼𝑦 = 𝜃(𝑌𝑒, 𝑌𝑐) 𝛼𝑧 = 𝜃(𝑍𝑒, 𝑍𝑐) (6)

El cálculo del ángulo entre dos vectores, ecuación (7), viene representado por la siguiente fórmula:

(7)

Con los ángulos calculados, podemos obtener la matriz de transformación, ecuación (8):

Ya hemos finalizado el proceso de construcción de nuestra matriz de transformación, el último paso, es aplicarla sobre los puntos de los *joints* de los distintos *frames* capturados, ecuación (9). La posición original viene representada por p y p’ es el punto resultante tras aplicar la transformación.

(9)

Antes de transformar el punto, es importante guardarnos la primera posición del primer *frame* de cada *joint.* Una vez aplicada la transformación sobre el punto, deberemos restarle las coordenadas del origen. Así pues, una forma de corroborar que estamos realizando correctamente la transformación, es observar si las coordenadas en el primer *frame* son 0,0,0 ello implicará que los *joints* en el primer *frame* los hemos establecido correctamente como el origen.

Hemos capturado la información de los 60 *frames* que conforman el gesto y lo hemos pasado a nuestro sistema de referencia. Sólo queda escribirlos en el *xml* haciendo uso de la serialización de la librería *Boost*. Una vez guardado las muestras que usaremos en nuestro modelo guardamos definitivamente al usuario en nuestra base de datos:



Figura 6.6. Registrar Usuario

En la figura anterior (Figura 6.6) cabe destacar una opción del sistema *SIRVAMED* llamada “Modo Carrusel”, esta opción está pensada para los pacientes que sufren un mayor índice de discapacidad, a tal punto que a lo sumo puedan realizar un único gesto (que tomaremos como aceptar). El modo Carrusel, emitirá la señal “siguiente” para navegar por la aplicación cada x segundos (los que se indiquen en la configuración del paciente) y el sistema sólo esperará a que el usuario realice el gesto de aceptar sobre el botón que tenga el foco.

Si el usuario ha realizado el entrenamiento del *Leap Motion* en la base de datos se guardará que dicho usuario desea utilizar la configuración personalizada y no la de por defecto.

Una vez registrado el paciente, sólo resta que se identifique para iniciar sesión en el sistema, momento en el que se cargarán los gestos del usuario identificado tanto del *Leap Motion*, como de la Kinect o del *Eye Tracker* (si es que han sido configurados, en caso contrario usará la configuración por defecto (Figura 6.7).

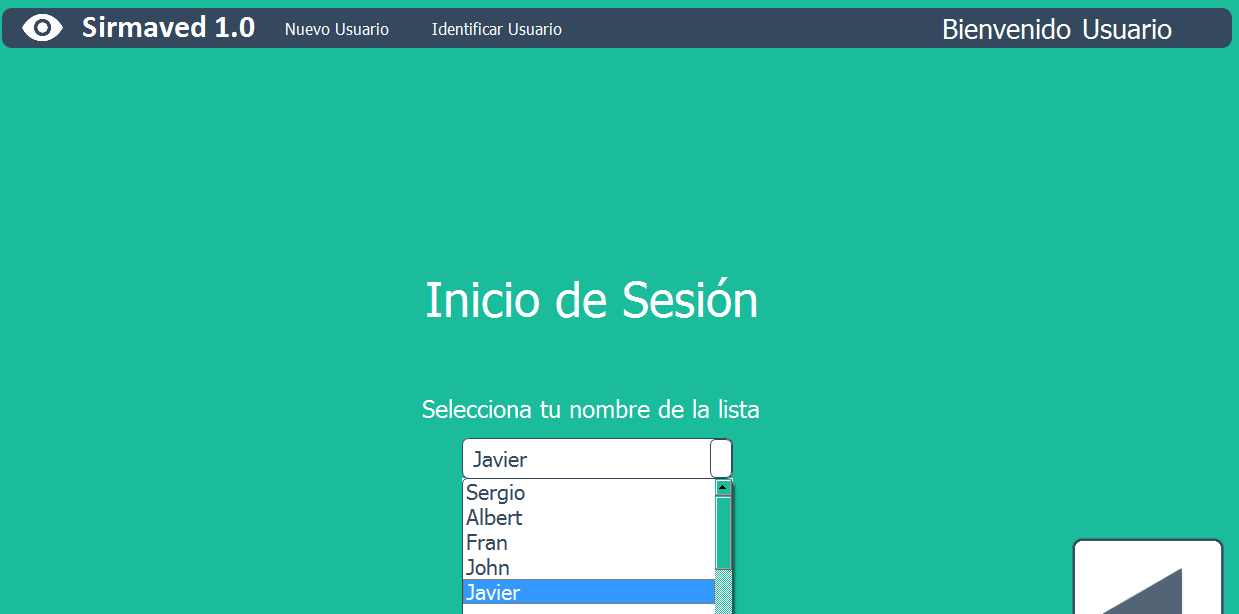


Figura 6.7. Identificar Usuario

## Reconocimiento de gestos

Cuando inicia sesión y se cargan los gestos (si ha realizado el entrenamiento), comenzamos en la ventana del menú principal (Figura 6.8).

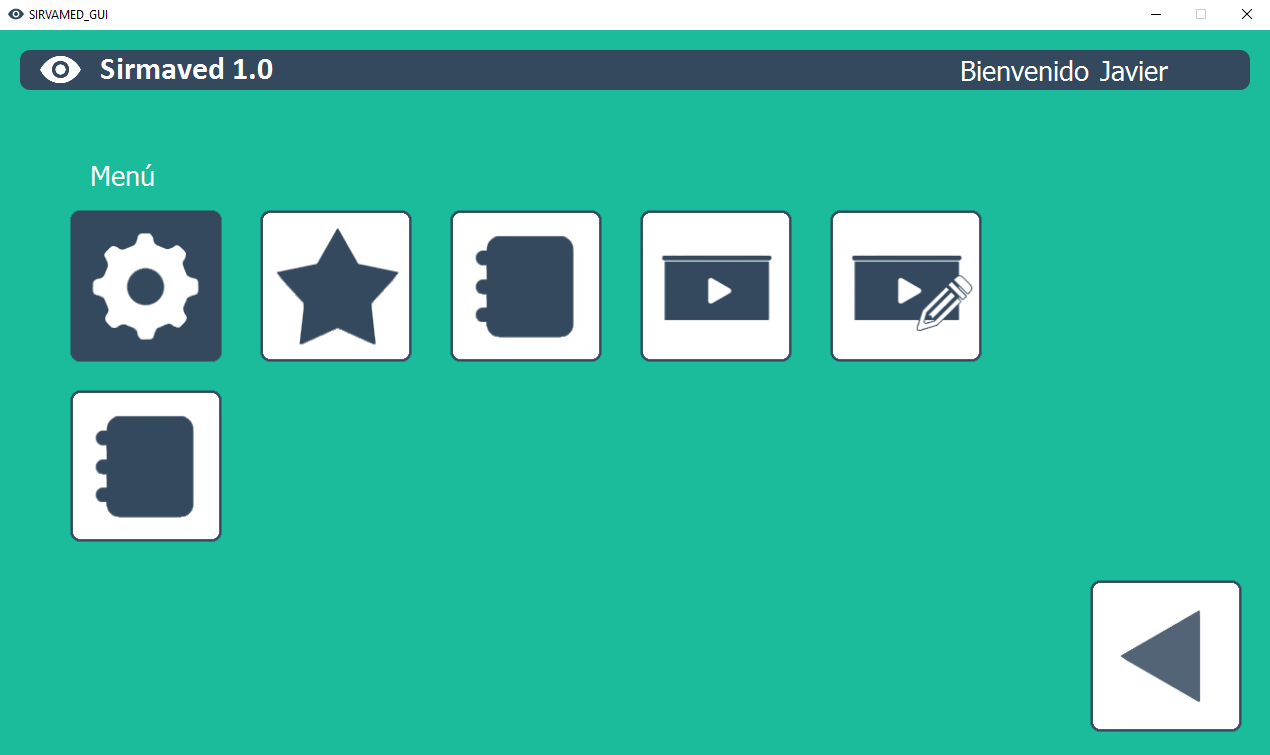
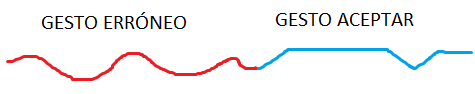


Figura 6.8. Menú Principal

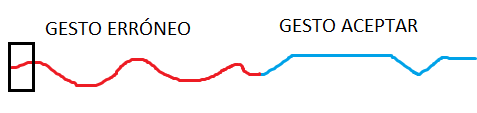
En este instante el usuario querrá navegar por el sistema y probar los juegos de estimulación cognitiva. Por lo que debemos detectar qué gesto está realizando. Para ello disponemos de un buffer circular que irá rellenándose con la información que extraigamos de los *frames* que nos envía el dispositivo. Si en el entrenamiento guardábamos 60 *frames,* nuestro buffer será un vector del mismo tamaño, de tal modo que si introducimos un elemento más a la ventana (el número 61), desplazará del vector el primer elemento. Iremos rellenando la ventana y comparando constantemente con los gestos cargados del usuario identificado.

Para vislumbrar mejor el funcionamiento de dicho buffer circular veremos el siguiente ejemplo:

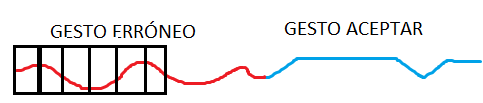
Vamos a realizar el gesto aceptar, pero antes de hacerlo, realizamos gestos “erróneos” que no representan ninguna acción. Luego hacemos el gesto correcto.



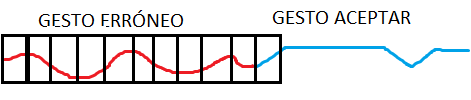
Al inicio de la ejecución del programa el buffer estará obviamente vacío, y lo iremos rellenando por cada *frame* que nos llegue.



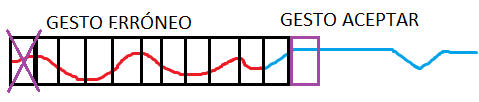
Aunque sólo tengamos un elemento en la ventana, lo comparamos con todos los gestos que tengamos (más adelante explicaremos cómo).



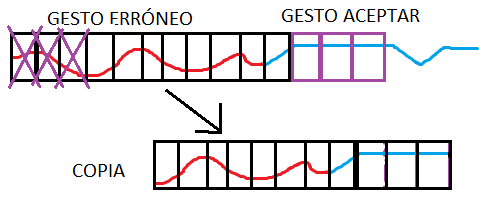
El proceso se repite, añadimos un elemento al buffer y comprobamos si el gesto almacenado con la información que tenemos hasta el momento corresponde con las muestras del modelo.



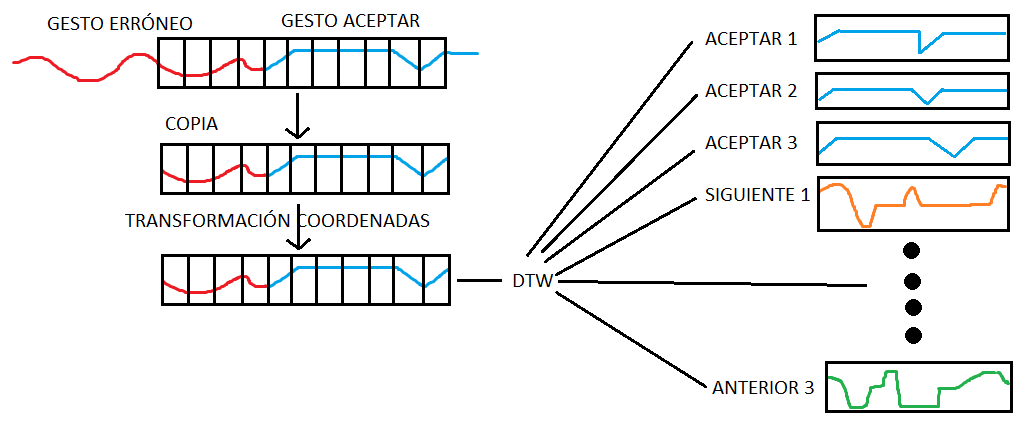
Cuando llenamos la ventana y aún no hemos detectado el gesto, seguimos añadiendo elementos y desplazando los primeros, que, al no haber creado el gesto con la ventana llena, sabemos que no corresponden a ninguna acción.



En algún momento la ventana se llenará con la información necesaria que corresponda al gesto correcto. Antes de comparar el contenido de la ventana con las muestras, dicho contenido tiene la información de sus coordenadas en relación al dispositivo como origen de coordenadas, por lo que también debemos realizar la trasformación de coordenadas del gesto que tengamos capturado para que sea “comparable” con las coordenadas transformadas en el entrenamiento. Dicha transformación, no podemos realizarla sobre el buffer circular, pues si transformamos la ventana original y en las siguientes iteraciones añadimos más información y volvemos a realizar la transformación, las coordenadas resultantes no serán correctas. Así pues, antes de comparar el contenido de la ventana, debemos hacer una copia en un vector a parte y sobre dicha copia aplicar el cambio de coordenadas.



Rellenamos el buffer de forma circular, desplazando los primeros elementos cuando se llene. Hacemos una copia del contenido de la ventana. Realizamos una transformación de coordenadas sobre la copia. Y aplicamos Dynamic Time Warping entre la copia transformada con las muestras:



Depende de la cota que pongamos al comparar, quizá no sea necesario completar el gesto del todo. Si es alta, aunque el gesto no sea idéntico, podrá devolverlo (pero ello implicará una mayor predisposición de detectar gestos erróneos). Si se ha detectado un gesto y hemos emitido la señal correspondiente, ya no nos interesa el contenido de la ventana (pues ya hemos identificado el gesto), así pues, la vaciamos y seguiremos rellenándola para detectar el siguiente gesto.

Sólo resta contemplar el algoritmo Dynamic Time Warping para hacer la comparación.

### Dynamic Time Warping

Explicamos el funcionamiento general del algoritmo DTW en la metodología, así como el pseudocódigo (Figura 5.8), pero hay varios aspectos a considerar. El algoritmo recibe dos parámetros, el gesto detectado en ejecución y el vector de gestos guardados del usuario.

Con DTW, lo que hacemos es comparar las trayectorias de los gestos y abstraernos de la velocidad con la que se realizan. Pero la “trayectoria de un gesto” debemos entender que es la trayectoria de los *joints* que conforman el gesto, es decir, las trayectorias de cada *joint* (en nuestro caso seis). Debemos aplicar DTW por cada joint. Por lo que procesaremos las entradas que recibimos para situar al mismo nivel el mismo *joint* de los 60 *frames* que forman el gesto. Por ejemplo, contar con un vector por cada *joint del* mismo tamaño del número de frames del gesto, y colocar en dicho vector toda la información de ese joint. Esto lo realizamos con el gesto recién creado, pero también con los guardados. Por cada gesto del usuario, procesamos los joints y aplicamos DTW por joint. Así comprobamos la trayectoria de cada joint. La distancia total del gesto, será la suma de distancias de cada *joint*.

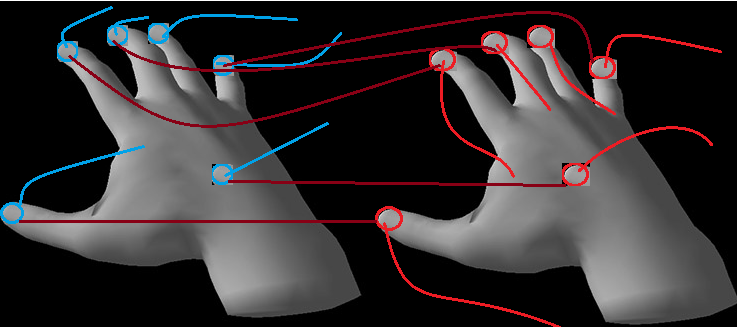


Figura 6.9. DTW entre joints [10]

Si aplicamos DTW cada vez que añadimos un elemento y tenemos tres muestras de cada gesto y lo ejecutamos por cada joint, esto contiene una carga computacional considerable. Por lo que añadimos dos pequeñas mejoras para acelerar el proceso. La primera mejora es guardarnos la distancia más pequeña que tengamos hasta el momento (la distancia entre el gesto capturado y el gesto que más se le parezca de momento) y si a la hora de comparar con otro gesto, éste tiene una distancia mayor, cortamos la comparación con dicho gesto. Es decir, si, por ejemplo, al comparar con el gesto “aceptar 1” (que es siempre el primero con el que se compara) nos devuelve una distancia de 8000 milímetros, nos guardamos esa distancia como la más pequeña hasta el momento (ya que la variable que almacena esa información se inicializa a infinito). Si al comparar con el gesto “aceptar 2”, y cuando vamos por el tercer *joint* la diferencia con ese gesto ya es 8500 milímetros, cortamos la comparación con “aceptar2”, pues ya sabemos que dicho gesto tiene una probabilidad menor de corresponderse con el gesto capturado y pasamos a comparar con el siguiente gesto. Es posible que el gesto con la menor distancia siga siendo una distancia considerable, por lo que la segunda mejora consiste en establecer una cota que límite la distancia máxima que estamos dispuesto a aceptar para que se corresponda con un gesto, por lo que si en medio de una comparación con un gesto, la diferencia es mayor a la menor acumulada hasta el momento o mayor que la cota, cortamos la comparación con la muestra y pasamos a la siguiente.

Si no se cumplen estas dos restricciones, devolverá un gesto no reconocido y el sistema no emitirá ningún evento. Si se ha reconocido, lanzamos la acción correspondiente. Entre cada acción dejamos un margen de un segundo para dar tiempo al usuario a posicionarse.

Si tenemos tres nuestras por gesto, para aplicar k-vecinos, sólo podemos quedarnos con k=1 o k=3, ya que debe ser impar para evitar empates. Con k=1, es el que mejor resultados da, por lo que devolvemos siempre el más cercano.

Con esto ya tenemos completado todo el proceso de reconocimiento de gestos: buffer circular que almacene la información ofrecida por el *Leap*, aplicar transformación de coordenadas y ejecutar DTW parta comparar con las muestras del entrenamiento del usuario.

## Aplicación de Rehabilitación: Sopa de Letras

El sistema cuenta con plantilla de juegos de estimulación cognitiva, diseñadas por diferentes miembros del equipo, como la siguiente (Figura 6.10. y 6.11) que consiste en encontrar las diferencias. Debemos memorizar las imágenes en un principio:



Figura 6.10. Ficha Estimulación Cognitiva

Y ser capaces de identificar la imagen que no aparecía antes.



Figura 6.11. Ficha Estimulación Cognitiva Respuesta

Este tipo de juegos, sirven como ayuda en la rehabilitación de los pacientes al obligarles a realizar un ejercicio de memorización.

El juego que he desarrollado consiste en una sencilla sopa de letras, haciendo uso de la librería Qt, se carga una imagen aleatoria y sus cinco respuestas asociadas (donde solo una es la correcta) al hacer el gesto siguiente o anterior cambiará de fila, y con el gesto aceptar sobre la fila que piense que es la correcta elegirá la fila donde aparece la respuesta correcta.

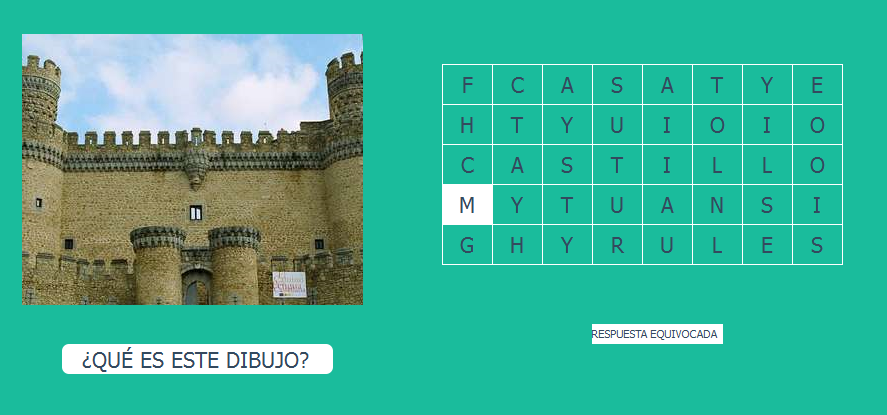


Figura 6.12. Sopa de letras

Hay varios ejemplos de imágenes para que el paciente se entretenga y reactive sus neuronas al intentar elegir la opción correcta de la sopa de letras. Todas estas páginas de estimulación se pueden jugar independientemente del dispositivo conectado.

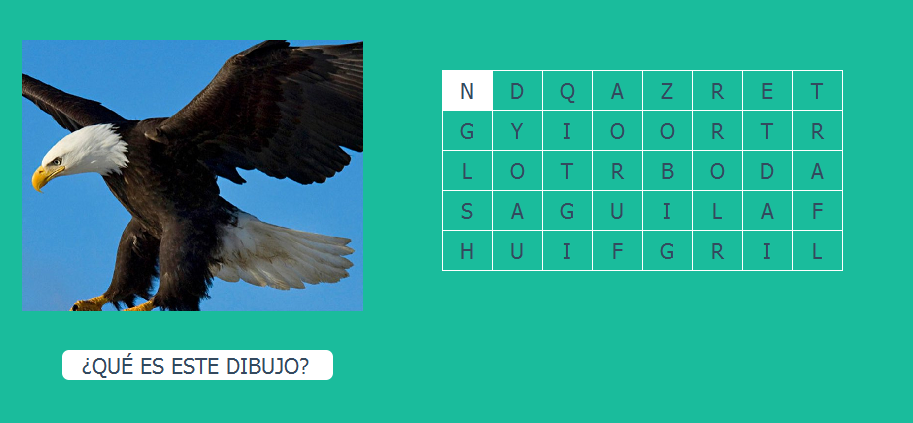


Figura 6.13. Sopa de letras2

## Análisis de Resultados

Una vez finalizado el desarrollo de nuestra aplicación, vamos a exponer y analizar los resultados obtenidos de nuestra implementación.

Hemos realizado pruebas con distintos usuarios y gestos sencillos, en primer lugar, ejecutamos el entrenamiento realizando los gestos que considere el paciente. Para evaluar la efectividad de las pruebas se han elaborado el mismo número de intentos con cada gesto guardado. Un dato a considerar es que el gesto que mejores resultados da (entorno al 90% de acierto) es el gesto de aceptar, el que la mayoría de usuarios hacen correctamente (esto es muy importante cuando se emplea el modo carrusel).

Podemos observar (Tabla 6.1) que la tasa media de acierto es del 80% mientras que el de equivocación es del 20%, donde se reparte un 6,7% de error (no reconoce ningún gesto) y un 13,3% de confusión donde detecta una acción que no corresponde con la que se ha realizado realmente. Los que acumulan el porcentaje de confusión son los de siguiente de anterior. Esto es debido a que los usuarios realizan el mismo gesto: mover la mano hacia un lado, generalmente en sentido anti horario para siguiente y horario para previo. Las personas, cuando realizan un gesto, la tendencia natural es posicional la mano en el punto de origen tras realizar el gesto: hacemos el gesto siguiente moviendo la mano a la izquierda, y la devolvemos a la posición original moviéndola a la derecha, ese acto, puede tomarse como anterior, lo mismo sucedería con un gesto que sea mover la mano hacia abajo y que otro gesto fuese moverla hacia arriba.

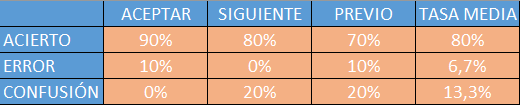
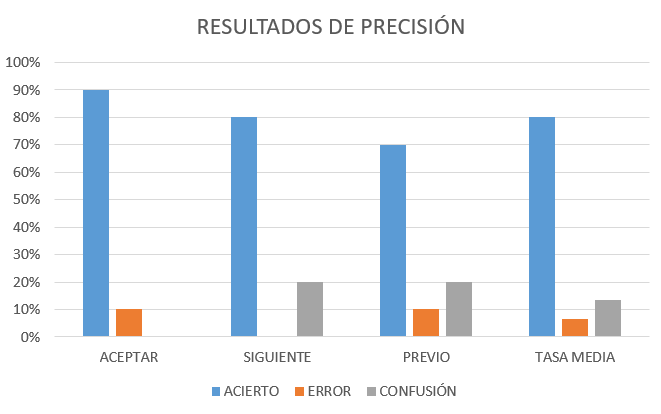


Tabla 6.1. Resultados de precisión

La siguiente gráfica (Gráfica 6.1) puede resumir el estado del sistema en el reconocimiento de gestos:



Gráfica 6.1. Resultados de precisión

A continuación, se realizaron pruebas para detectar gestos grabados previamente por otros usuarios, para comprobar cómo de fiable es el sistema entre distintos pacientes. Los resultados demuestran que la precisión desciende drásticamente cuando un usuario prueba los gestos grabados por otros, excepto el de aceptar que da unos resultados de precisión muy parecidos (Tabla 6.2).

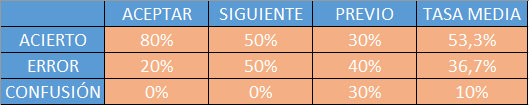


Tabla 6.2. Resultados con gestos grabados por otro usuario

La precisión baja a poco más del 50%, donde casi todo el porcentaje de acierto reside prácticamente en el gesto aceptar. Para el gesto de siguiente de cada diez intentos reconoce cinco y otros cinco no detecta nada. Y con previo la precisión es donde más cae (un 70% de equivocación).

¿A qué es debido esto? A diversos factores relacionados con el *Leap Motion*. El *Leap Motion*, nos devuelve las coordenadas según la mano que detecte, es decir, una mano más grande o más pequeña que la que se utilizó para el entrenamiento tendrá unas coordenadas diferentes. Por ejemplo, la posición general de una falange, si el dedo es más largo, tendrá unas coordenadas diferentes. Sucede otro añadido que baja la precisión y es exactamente la precisión del dispositivo, como podemos ver en la siguiente imagen (Figura 6.14): Claramente posicionamos la mano derecha recta, y el dispositivo representa la izquierda, retorcida sobre la muñeca y arqueando los dedos.

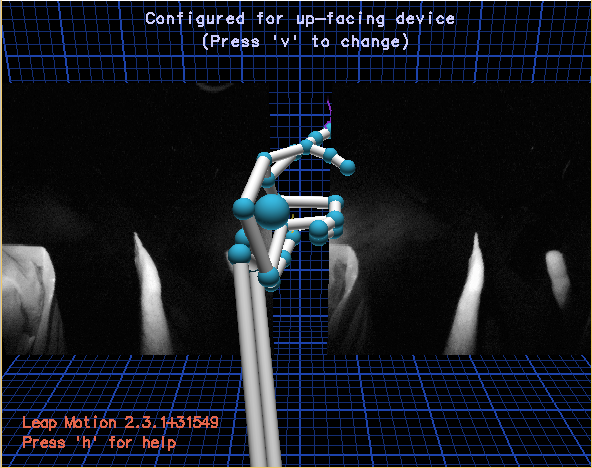
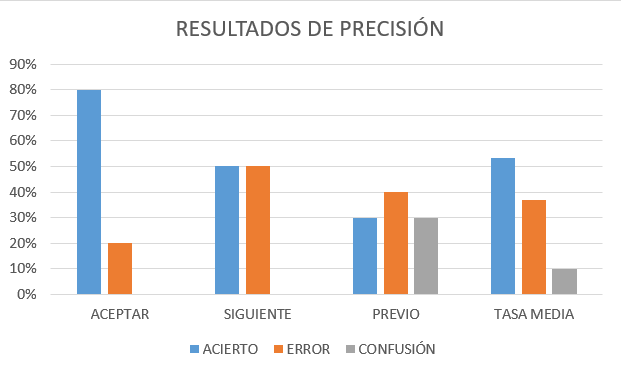


Figura 6.14. Precisión del Leap Motion

*Leap Motion* hace uso de dos cámaras y realiza una unión de la información que éstas capturan, en ocasiones es muy buena, pero en otras no tanto. Si encima, el dispositivo tiene alguna mancha, aunque sea diminuta, comenzará a realizar gestos extraños que no se corresponden en absoluto con el realizado. Puede que en el entrenamiento grabemos un gesto, y cuando lo efectuamos a la hora de comparar no lo detecte, porque es posible que el gesto que detectó en el entrenamiento fuese como en la imagen anterior: una mano retorcida.

En la siguiente gráfica (Gráfica 6.2) podemos vislumbrar mejor los resultados, el porcentaje medio de confusión entre gestos se reduce considerablemente (hasta un 10%) únicamente por que el porcentaje de error, en el que no reconoce ningún gesto, se eleva hasta más del 30%, y no se incrementa aún más gracias al gesto sencillo y poco confuso de aceptar.



Gráfica 6.2. Resultados con gestos grabados por otro usuario

Recordemos que utilizamos seis *joints*, si el dispositivo nos los deforma todos, la palma y los dedos, al realizar DTW, la distancia será enorme y no devolverá ningún gesto. Hay que tener cuidado con la precisión del dispositivo. Los resultados de las gráficas 6.1 y 6.2, se realizaron en todo momento con el visualizador abierto para cerciorarnos de que captaban correctamente la información. Por lo que, en las ocasiones que captaba una mano deformada, no fueron incluidas en los % de precisión.

# Conclusiones

Hemos visto todos los pasos necesarios para realizar tanto la detección como el reconocimiento de gestos: elegir joints representativos de la mano que identifiquen un gesto. Disponer de muestras suficientes para poder compararlas más adelante. Realizar una transformación de coordenadas para que las posiciones no dependan del dispositivo como origen de referencia. Y aplicar un algoritmo, en nuestro caso Dynamic Time Warping, para comparar los gestos, así como la necesidad de buscar restricciones para acelerar el proceso, sobre todo en nuestro caso, donde contamos con seis *joints*, lo que supone una carga de cómputo considerable.

Repasando los objetivos técnicos que planteamos, hemos logrado desarrollar un sistema que permite a los pacientes utilizarlo con cualquier dispositivo y ofrecer la posibilidad de realizar gestos por defecto o personalizados, donde se guardan correctamente en la base datos toda la información de los usuarios. Y desarrollar un juego, la sopa de letras, para que los pacientes puedan probarlo mientras el entorno detectar los gestos realizado a la hora de navegar por las ventanas.

A la luz de los resultados obtenidos hay que ser precavidos con la precisión del dispositivo y asegurarnos de que esté en buen estado, evitando lo máximo posible la suciedad ya que una simple marca de yema de dedos entorpecerá la captura de información de sus cámaras. También hemos deducido que, con distintos pacientes sobre el mismo entrenamiento, los resultados se reducen, ya que el dispositivo ofrece coordenadas distintas según la mano que aparezca sobre el dispositivo. Según podemos observar en la página oficial de Leap Motion [1], a causa de esta baja precisión, la empresa lleva cierto tiempo abandonando su dispositivo para centrarse en las gafas de Realidad Virtual, las cuales, si bien usan la misma tecnología que su predecesor, buscan solventar el problema de la precisión.

Por último, con respecto a los objetivos personales, se ha logrado adquirir las capacidades de trabajar en conjunto junto a un verdadero equipo de desarrollo y aprender todas las tecnologías que forman el sistema*, Leap Motion, Qt* e incluso los fundamentos de la *Kinect* y el *Eye Tracker*. Y la más importante, haber logrado desarrollar un sistema que sirva en la rehabilitación de pacientes con daño cerebral.

# Bibliografía

[1] Página Oficial Leap Motion.

<https://www.leapmotion.com/>

consultado en fecha 15/01/2016

[2] Página Oficial Tobii EyeX.

http://www.tobii.com/xperience/

consultado en fecha 07/11/2015

[3] Página de Microsoft sobre Kinect

<https://developer.microsoft.com/en-us/windows/kinect>

consultado en fecha 14/11/2015

[4] Lengua de señas

<https://es.wikipedia.org/wiki/Lengua_de_se%C3%B1as>

consultado en fecha 21/04/2016

[5] Reconocimiento de Gestos

<https://es.wikipedia.org/wiki/Reconocimiento_de_gestos>

consultado en fecha 21/04/2016

[6] Leap Trainer.js

<https://www.youtube.com/watch?v=JVqalPM9pHs>

consultado en fecha 02/12/2015

[7] Recognizing Schaeffer's gestures for robot interaction

<http://simd.albacete.org/actascaepia15/papers/01045.pdf>

consultado en fecha 13/12/2015

[8] Gesture recognition using RFID technology

<https://pdfs.semanticscholar.org/8d02/95c3281b281e0a8bf27e6d834543035fd6fe.pdf>

consultado en fecha 21/04/2016

[9] Informe de ADACE-CLM sobre el daño cerebral en España

<http://www.adaceclm.org/DocumentosInteres/Documentosda%C3%B1ocerebral/Documents/Magnitud%20del%20Da%C3%B1o%20Cerebral%20Sobrevenido%20en%20Espa%C3%B1a%20y%20Castilla-La%20Mancha%202013.pdf>

consultado en fecha 15/11/2015

[10] Fuente del modelo 3D para realizar la foto de comparación de joints con DTW

<https://www.creativecrash.com/>

consultado 05/04/2016

[11] Fuente de la imagen de un modelo basado en el esqueleto

<http://prog3.com/sbdm/blog/zouxy09/article/details/8161617>

consultado 01/02/2016