Universidad Zaragoza

Proyecto Final de Carrera

Ingeniería de Telecomunicación

Reconocimiento de secuencias gestuales adquiridas con Kinect utilizando HMMs

Autor

Victoria Carmona Balfagón

Director

Antonio Miguel Artiaga

Escuela de ingeniería y arquitectura
Universidad de Zaragoza

Curso 2016/2017

A todos aquellos que han hecho posible
que este proyecto vea finalmente la luz.
A mi familia, mis compañeros y amigos,
gracias de corazón por todo su apoyo

RESUMEN

Kinect es el accesorio de las videoconsolas más modernas que permite jugar a videojuegos usando sólo el cuerpo, sin ningún tipo de control. Con su conjunto de sensores es posible jugar como en la vida real, pues las siluetas de los jugadores son capturadas por dichos sensores para posteriormente ser emparejadas con un modelo de esqueleto. En este proyecto, se implementa un reconocedor de secuencias gestuales utilizando el sensor Kinect. Con el kit de desarrollo de software liberado por Microsoft, grabamos diferentes movimientos en ficheros de texto creando una base de datos con la que trabajaremos durante el desarrollo del proyecto.

En primer lugar, construimos el reconocedor en Matlab. Para ello, es necesario realizar el entrenamiento de los modelos ocultos de Markov (HMMs), obteniendo la secuencia óptima de estados con el algoritmo de Viterbi y reestimando los parámetros en cada una de las iteraciones (método de Baum-Welch). Una vez calculados los HMMs, implementamos la función de reconocimiento obteniendo el gesto que proporcione mayor verosimilitud. El siguiente paso, es implementar el sistema en lenguaje C++ que es el lenguaje de programación usado en el SDK de Kinect.

Por último, se implementa una aplicación con distintos juegos de identificación de movimientos cuya base será el reconocedor construido. Se trata de una herramienta útil para los terapeutas que trabajan con niños con movilidad reducida y problemas de interacción. Dispone de varias opciones de configuración y un apartado de entrenamiento que permite crear HMMs de diferentes movimientos. La aplicación contiene 4 juegos diferentes: Gestos, Adivinanzas, Frases y Evocación que podrán ayudar a los niños a mejorar su capacidad motora y coordinación, así como su capacidad de razonamiento al relacionar las distintas imágenes con sus movimientos.

Contents

Chapter 1

Introducción

1.1 Objetivos del proyecto

Este proyecto se enmarca dentro de una línea de realización de herramientas de asistencia a la terapia que sigue la línea de proyectos anteriores que ya desarrollaron este tipo de herramientas dentro del grupo Vivolab[1] en colaboración con Colegio Público de Educación Especial Alborada (CPEE Alborada). El propósito de este proyecto es utilizar el dispositivo Kinect para construir un reconocedor de secuencias gestuales que pueda ayudar a niños con movilidad reducida y problemas de interacción, teniendo como objetivo la comunicación y la terapia.

Los objetivos de este proyecto son, en primer lugar, usar la cámara del Kinect para grabar distintos movimientos y crear una base de datos con la que trabajaremos para avanzar en el desarrollo del proyecto y poder medir resultados cuantitativos de tasas de acierto y error. Se captarán 20 puntos del esqueleto cuyas coordenadas cartesianas grabaremos en ficheros para un número controlado de movimientos simples, que posteriormente trataremos de identificar. Estos, serán cargados en Matlab donde podremos representar los movimientos del esqueleto. A partir de ellos, se extraerán distintos ángulos del cuerpo que serán los datos de entrada a nuestro reconocedor basado en modelos ocultos de Markov (HMMs).

En segundo lugar, se realizará un entrenamiento iterativo de los modelos ocultos de Markov, para ello, se dispondrá de varios ficheros con los datos del movimiento de cada modelo capturados en la base de datos. Una vez tengamos los modelos de cada movimiento, se construirá el reconocedor utilizando el algoritmo de Viterbi y se evaluarán los ficheros de test de nuestra base de datos con cada uno de los modelos obtenidos en la fase de entrenamiento.

[1] `www.vivolab.es`

Por último, se implementará el sistema en lenguaje C++ que es el lenguaje de programación usado en el SDK de Kinect. De esta forma, se desarrollará una aplicación con distintos juegos de identificación de movimientos cuya base será el reconocedor construido. Como el objetivo de estos juegos es ayudar a niños a mejorar su capacidad de movimiento y razonamiento, se utilizarán dibujos y sonidos que capten su atención. A través del juego, los niños podrán practicar los movimientos y asociar imágenes con sus gestos.

1.2 Estado del arte

El avance en el reconocimiento de gestos ha sido muy significativo en los últimos años. Se lleva a cabo mediante cámaras que capturan las imágenes y el posterior procesado de la información por medio de algoritmos matemáticos.

La primera interfaz gestual que se comercializó fue el dispositivo Kinect en el año 2010. Su tecnología se compone de una cámara y un sensor de profundidad, que mide mediante infrarrojos el relieve de los objetos. El dispositivo analiza y detecta un espacio definido en 3 dimensiones, de tal manera que es capaz de seguir los movimientos de los jugadores, emparejando sus siluetas con un modelo de esqueleto.

El dispositivo Kinect se ha hecho muy popular al conectarse a la consola Xbox, permitiendo así, jugar a videojuegos usando sólo el cuerpo. Sin embargo, el reconocimiento de gestos va más allá de los juegos, nos permite comunicarnos con las máquinas e interactuar con ellas sin necesidad de dispositivos mecánicos. El kit de desarrollo de software liberado por Microsoft, permite a los usuarios crear sus propias aplicaciones de control del movimiento utilizando el sensor Kinect[2]. Enfoques actuales en el campo incluyen el uso de esta tecnología para reconocer movimientos [14] [15].

En este proyecto se construye un reconocedor de gestos utilizando el sensor Kinect y así desarrollar un herramienta que sirva de apoyo a la terapia. Este tipo de sistemas ha sido siempre muy demandado por parte de profesionales y terapeutas, resultando útil para la rehabilitación física de pacientes con discapacidad motora [13] [16]. La herramienta construida en este proyecto servirá de apoyo para mejorar la capacidad motora de los niños con movilidad reducida y problemas de interacción.

En el ámbito de la Educación Especial, el grupo Vivolab de la univer-

[2]Kinect para Windows `https://dev.windows.com/en-us/kinect`

sidad de Zaragoza lanzó la herramienta "Vocaliza"[3], una aplicación para mejorar los problemas en el desarrollo del lenguaje en niños con alteraciones en el habla [6].

Otras investigaciones actuales relacionadas con el reconocimiento de gestos, se centran en reconocer el lenguaje de signos [8] [9]. La implementación de estos sistemas requiere la detección de la manos [10] y la observación de los gestos realizados, extrayendo sus caraterísticas [11]. En [12] se lleva a cabo el reconocimiento de los gestos de las manos utilizando Modelos ocultos de Markov (HMMs), que es el método elegido para construir el reconocedor en este proyecto.

1.3 Metodología de trabajo

A continucación, se describen las distintas fases que se han llevado a cabo para la realización de este proyecto.

1.3.1 Documentación

La primera fase de este proyecto consistió en la lectura de textos científicos basados en el reconocimiento del lenguaje de signos. De esta forma, se conoció el estado del arte y las distintas técnicas utilizadas para la detección de los movimientos y su reconocimiento.

También se realizó un estudio de los modelos ocultos de Markov con funciones de densidad de probabilidad continuas (CDHMMs) ya que con ellos se han obtenido buenos resultados en aplicaciones de reconocimiento de secuencias de patrones. Esto nos lleva a estudiar el entrenamiento de los modelos y el algoritmo de Viterbi. [1][2]

El siguiente paso fue la lectura de la documentación asociada al SDK de Kinect[4] para conocer su funcionamiento y la forma de desarrollar aplicaciones de control del movimiento.

1.3.2 Construcción del reconocedor

Al tratarse de un modelo estadístico con muchos parámetros que estimar, se necesita disponer de datos de entrenamiento. Para ello, primero se crea una base de datos grabando distintos movimientos con la cámara de Kinect. Se guardan las coordenadas cartesianas de los 20 puntos del esqueleto en

[3]Proyecto Comunica `http://dihana.cps.unizar.es/~alborada/herramientas.html`
[4]Kinect para Windows `https://dev.windows.com/en-us/kinect`

ficheros de texto que serán leídos con una función de Matlab. Posteriormente se procesan estos datos y se calculan distintos ángulos del cuerpo que servirán para reconocer los movimientos.

A continuación, se construyen los modelos ocultos de Markov en Matlab. Lo primero que se necesita es una estructura para almacenar los parámetros de los HMMs, la matriz de transición y la función de probabilidad de la secuencia de salida que se representa con una mezcla de Gaussianas por estado. En segundo lugar, implementamos el algoritmo de Viterbi con el que se obtendrá la secuencia óptima de estados. Para entrenar los HMMs, se reestiman sus parámetros calculando la máxima verosimilitud en varias iteraciones hasta que dicho valor no cambie de forma significativa. De esta forma, sabremos cual es el número óptimo de iteraciones que debemos realizar para construir nuestro reconocedor. Se representa la matriz de probabilidades junto con la mejor secuencia de estados en cada iteración, comprobando como la máxima verosimilitud se adapta al mejor camino conforme se avanza en el entrenamiento.

Para comprobar las prestaciones del sistema, una vez tenemos los modelos de varios movimientos, se reconocen los ficheros de test utilizando el algoritmo de Viterbi. Se evalúa cada fichero con cada uno de los modelos construidos dando como resultado áquel movimiento con el que obtenemos mayor verosimilitud. Se analizan 7 movimientos obteniendo buenos resultados en el reconocimiento.

Por último, se implementa el sistema en lenguaje C++ siguiendo los mismos pasos llevados a cabo en Matlab. Se analizan las prestaciones con cuatro usuarios diferentes calculando tasas de acierto y error.

1.3.3 Diseño y desarrollo de la aplicación

La aplicación se diseñó como herramienta de ayuda a los terapeutas que tratan a niños con problemas de movilidad e iteracción con el medio que les rodea. Por tanto, el entorno gráfico debe ser simple utilizando imágenes coloridas que capten la atención de los niños. Con los distintos juegos, se podrán practicar diferentes movimientos de forma divertida ayudando a mejorar la capacidad motora.

Para que la aplicación sea versátil, se desarrollan distintas opciones de configuración en los juegos, como el nivel de dificultad, los movimientos a realizar o las imágenes, adivinanzas y frases que se van a utilizar. También se adapta a distintos usuarios ya que se puede realizar el entrenamiento de los modelos con los movimientos del niño que vaya a jugar con la aplicación.

En el capítulo 3 de esta memoria se describen las características de los distintos juegos de la aplicación, así como el diseño y desarrollo de los mismos.

1.4 Contenidos de la memoria

La organización de la memoria es la siguiente:

- Capítulo 1: introducción al proyecto fin de carrera, describiendo los objetivos del mismo y las distintas tareas realizadas. También se resumen algunos enfoques actuales en el reconocimiento de los movimientos.
- Capítulo 2: explica la base teórica de la construcción del reconocedor: Modelos Ocultos de Markov HMMs, algoritmo de Viterbi y algoritmo de Baum-Welch. En el último apartado, se analizan las prestaciones del reconocedor construido.
- Capítulo 3: describe el diseño y la estructura de la aplicación. Se explica cómo realizar el entrenamiento y configurar los distintos juegos. También se detalla el desarrollo llevado a cabo al construir la aplicación.
- Capítulo 4: presenta las conclusiones obtenidas, así como las posibles líneas futuras de investigación.
- Apéndice A: contiene las estructuras definidas y las funciones desarrolladas en lenguaje C++ para construir el reconocedor. Se detalla la implementación de la interfaz de usuario utilizando las funciones API de Windows. Por último se resume el funcionamiento del sensor Kinect.

Chapter 2

Reconocedor de secuencias gestuales

2.1 Reconocimiento de gestos con HMMs

Cada gesto se define como la secuencia de símbolos $O = o_1, o_2, ...o_T$ donde o_t representa las coordenadas del esqueleto en el instante de tiempo t mediante un vector.
Considerando un conjunto de movimientos m_i, el gesto reconocido será aquel con probabilidad máxima dada la secuencia de símbolos O,

$$Gesto\ reconocido = \arg\max_i P(m_i|O) \tag{2.1}$$

Para calcular esta probabilidad, se utiliza la regla de Bayes,

$$P(m_i|O) = \frac{P(O|m_i)P(m_i)}{P(O)} \tag{2.2}$$

$P(O)$ es la misma para todos los movimientos, por tanto no hace falta tener en cuenta este término para conocer cuál de ellos tiene mayor probabilidad. Considerando también que todos los movimientos tienen la misma probabilidad $P(m_i)$, maximizar $P(m_i|O)$ es lo mismo que maximizar $P(O|m_i)$. Para calcular esta probabilidad se utilizan modelos ocultos de Markov.

Un modelo oculto de Markov o HMM es una máquina de estados finitos que cambia de estado con el tiempo y genera una secuencia de símbolos de salida con una función de probabilidad $b_j(o_t)$ dependiente del estado. La secuencia de estados por los que pasa el modelo es desconocida u oculta. Considerando la propiedad de Markov, la probabilidad de encontrarse en un estado en un instante de tiempo determinado t, solo depende del estado en que estuviese en el instante anterior $t-1$.

La matriz de las probabilidades de transación entre estados se define

$$A = \{a_{ij}\} \text{ siendo } a_{ij} = P(q_t = j | q_{t-1} = i) \tag{2.3}$$

La topología de la matriz de transición va a ser de un tipo especial llamado *left-to-right*, donde las transiciones solo son posibles entre el propio estado y el siguiente:

$$A = \begin{pmatrix} a_{11} & a_{12} & 0 & \dots & 0 \\ 0 & a_{22} & a_{23} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & a_{NN} \end{pmatrix} \tag{2.4}$$

En este proyecto, la función de probabilidad de la secuencia de salida se modela con una mezcla de Gaussianas:

$$b_j(o_t) = \sum_{k=1}^{K} c_{jk} N(o_t; \mu_{jk}, \Sigma_{jk}) \tag{2.5}$$

$$N(o; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n |\Sigma|}} e^{-\frac{1}{2}(o-\mu)^T \Sigma^{-1} (o-\mu)} \tag{2.6}$$

donde K es el número de componentes, c_{jk} son los pesos de las Gaussianas, μ_{jk} son las medias y Σ_{jk} las varianzas.

En la figura 2.1 se muestra un ejemplo de HMM moviéndose por la secuencia de estados $Q = 1, 2, 2, 3$ y generando la secuencia de salida o_1 a o_4. La probabilidad de generar dicha secuencia de salida O dado el modelo de Markov M, la secuencia de estados Q, es igual a:

$$P(O, Q|M) = \pi_1 b_1(o_1) a_{12} b_2(o_2) a_{22} b_2(o_3) a_{23} b_3(o_4) \tag{2.7}$$

siendo π_1 la probabilidad inicial del estado 1, $b_j(o_t)$ la probabilidad de salida del símbolo o_t en el estado j y a_{ij} la probabilidad de transición entre los estados i y j.

En el reconocedor que estamos construyendo, la secuencia de salida son los valores que toman los distintos ángulos del cuerpo al realizar el gesto $O = o_1, o_2, ...o_T$. La secuencia de estados es desconocida, por lo que la probabilidad de generar una salida dado un modelo de Markov se calcula como la suma de todas las posibles secuencias de estados $Q = q(1), q(2), ...q(T)$.

$$P(O|M) = \sum_{Q} a_{q(0)q(1)} \prod_{t=1}^{T} b_{q(t)}(o_t) a_{q(t)q(t+1)} \tag{2.8}$$

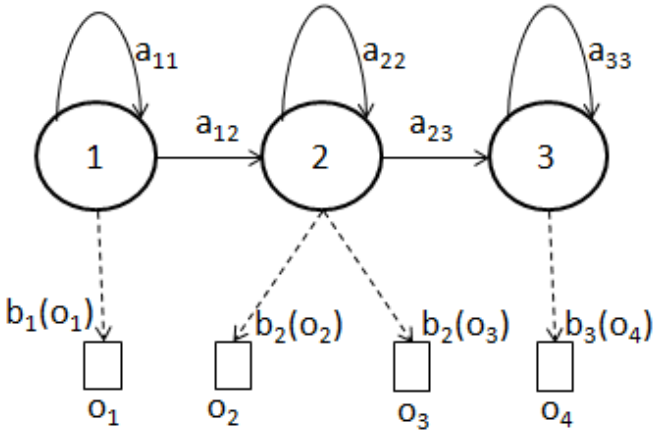


Figure 2.1: Ejemplo de Modelo Oculto de Markov (HMM) moviéndose por la secuencia de estados $Q = 1, 2, 2, 3$ y generando la secuencia de salida o_1 a o_4.

Considerando la secuencia de estados más probable, la ecuación 2.8 se aproxima en la práctica como:

$$\hat{P}(O|M) = \max_{Q} \left\{ a_{q(0)q(1)} \prod_{t=1}^{T} b_{q(t)}(o_t) a_{q(t)q(t+1)} \right\} \tag{2.9}$$

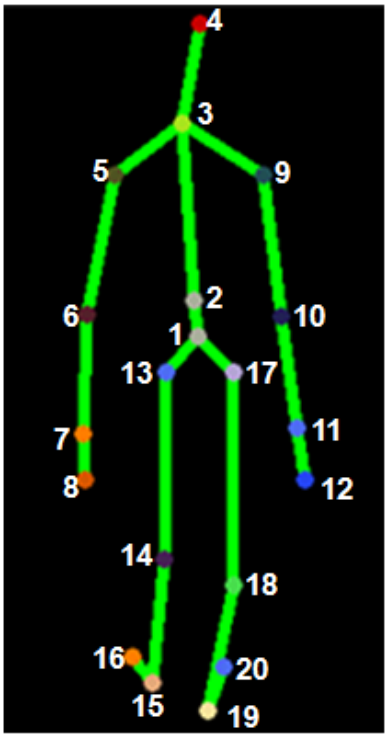
Por razones de precisión numérica, vamos a utilizar la ecuación 2.9 en logaritmo, quedando:

$$\hat{P}(O|M) = \max_{Q} \left\{ \log a_{q(0)q(1)} + \sum_{t=1}^{T} \log b_{q(t)}(o_t) + \log a_{q(t)q(t+1)} \right\} \tag{2.10}$$

Cada movimiento m_i se representa con el modelo oculto de Markov M_i, por lo que $P(O|m_i) = P(O|M_i)$. Por tanto, conociendo los parámetros $\{a_{ij}\}$ y $\{b_j(o_t)\}$ de cada modelo M_i, podemos resolver la ecuación 2.1 y reconocer el gesto realizado. La estimación de los parámetros de cada HMM se realiza entrenando cada modelo con varias muestras del movimiento correspondiente.

2.2 Datos de entrada del reconocedor

Para crear una base de datos con la que trabajar en la construcción del reconocedor de secuencias gestuales, grabamos distintos movimientos con la cámara del Kinect. Para ello, utilizamos el kit de desarrollo de software liberado por Microsoft que almacena la posición (x, y, z) de 20 puntos del esqueleto en una estructura de datos (figura 2.2).



1	Centro cadera
2	Columna
3	Hombro centro
4	Cabeza
5	Hombro izquierdo
6	Codo izquierdo
7	Muñeca izquierda
8	Mano izquierda
9	Hombro derecho
10	Codo derecho
11	Muñeca derecha
12	Mano derecha
13	Cadera izquierda
14	Rodilla izquierda
15	Tobillo izquierdo
16	Pie izquierdo
17	Cadera derecha
18	Rodilla derecha
19	Tobillo derecho
20	Pie derecho

Figure 2.2: Imagen del esqueleto capturado por Kinect identificando los 20 puntos del cuerpo cuyas coordenadas (x, y, z) son almacenadas en una estructura de datos.

Para la duración de cada gesto consideramos 2.5 segundos que siendo la tasa de captura de Kinect de $20fps$, corresponde a 50 capturas con la posición de los 20 puntos del esqueleto. Guardamos esta información en una matriz de dimensiones 50x60 (50 filas, una por cada captura de posiciones y 60 columnas - 20 puntos del esqueleto x 3 coordenadas). Para procesar estos movimientos en Matlab, creamos ficheros de texto con cada uno de los gestos grabados con Kinect.

Creamos la función *read_mov.m* en Matlab para leer los ficheros de texto con las posiciones de los puntos del esqueleto. Para representar el movimiento necesitamos una matriz de 20 filas (una por cada punto) y 3 columnas (una por cada coordenada x, y, z). Por tanto, transformamos cada una de las filas del fichero en una matriz de dimensiones 20x3. Representamos las coordenadas x e y de cada una de las capturas y con la función *movie.m* de Matlab vemos el movimiento. En la figura 2.3 se muestran 3

instantes de tiempo ($t = 0s, t = 2s, t = 4s$) del video obtenido al grabar un movimiento levantando el brazo derecho y después el brazo izquierdo.

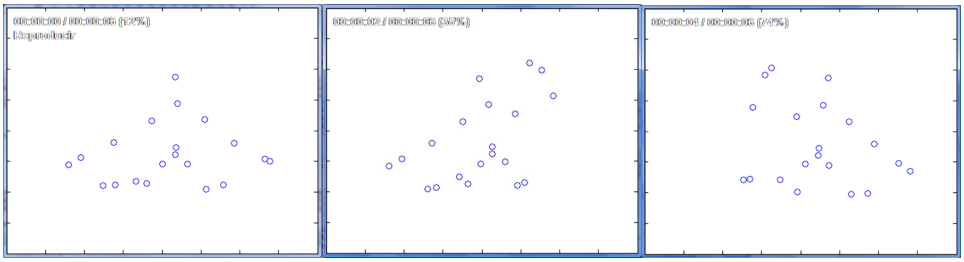


Figure 2.3: Capturas del video obtenido con la función *read_mov.m* en los instantes de tiempo $t = 0s, t = 2s, t = 4s$ al grabar un movimiento levantando el brazo derecho y después el brazo izquierdo

A partir de estos ficheros de texto con los movimientos grabados con la Kinect, extraemos distintos ángulos del cuerpo que serán los datos de entrada a nuestro reconocedor basado en modelos ocultos de Markov (HMMs). Para ello, obtenemos los vectores de las 2 líneas que forman el ángulo a calcular, como la diferencia entre las coordenadas del punto final y el inicial:

$$\vec{PQ} = Q - P = (q_1 - p_1)x + (q_2 - p_2)y + (q_3 - p_3)z \tag{2.11}$$

Una vez obtenidos los 2 vectores, calculamos el ángulo que forman con la ecuación 2.12

$$\alpha = \arccos \frac{\vec{u}.\vec{v}}{|\vec{u}|.|\vec{v}|} = \arccos \frac{x_1.x_2 + y_1.y_2 + z_1.z_2}{\sqrt{x_1^2 + y_1^2 + z_1^2}.\sqrt{x_2^2 + y_2^2 + z_2^2}} \tag{2.12}$$

En el desarrollo de este proyecto se van a utilizar los 8 ángulos mostrados en la figura 2.4. Para el reconocimiento de otros movimientos, por ejemplo un movimiento con las piernas, será necesario añadir los ángulos correspondientes a la función *read_mov.m*. Cambiando el número de dimensiones y agregando una nueva fila a la matriz con los valores del nuevo ángulo, se podrá utilizar el reconocedor construido para identificar cualquier tipo de movimiento.

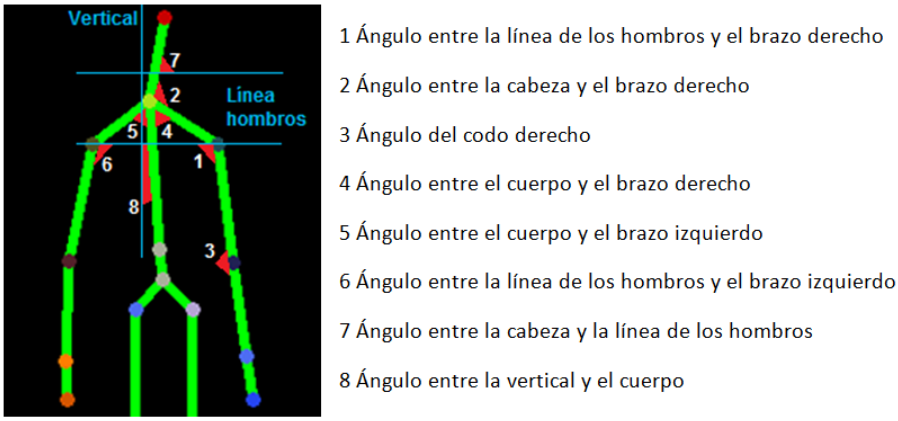


Figure 2.4: Imagen del esqueleto capturado por Kinect identificando los 8 ángulos del cuerpo utilizados en el desarrollo del proyecto

2.3 Entrenamiento de modelos ocultos de Markov

El entrenamiento consiste en la reestimación de los parámetros de los HMM realizando varias iteraciones. Para ello, empleamos varios ficheros con los datos del movimiento de cada modelo capturados con la cámara del Kinect.

2.3.1 Secuencia óptima de estados (Algoritmo de Viterbi)

Para encontrar la secuencia óptima de estados utilizamos el algoritmo de Viterbi. Se define la variable $\delta_t(i)$ como la probabilidad del mejor camino hasta el estado i en el instante de tiempo t para la secuencia de observaciones $o_1, o_2, ...o_t$:

$$\delta_t(i) = \max_{q_1,q_2,...,q_{t-1}} P(q_1, q_2, ..., q_{t-1}, q_t = i, o_1, o_2...o_t|M) \tag{2.13}$$

Esta función se calcula para todos los estados e instantes de tiempo de forma iterativa.

$$\delta_{t+1}(j) = \max_{1\leq i\leq N}\Big[\delta_t(i)(a_{ij})\Big]b_j(o_{t+1}) \tag{2.14}$$

siendo a_{ij} la probabilidad de transición del estado i al j y N el número de estados. Como se ha comentado en el apartado 2.1 de esta memoria, las transiciones solo serán posibles entre el propio estado y el siguiente (matriz *left-to-right* 2.4) y ambas se inicializan con la misma probabilidad ($a_{ii} = 0.5$ y $a_{ii+1} = 0.5$).

En el instante inicial $t = 1$, la función es igual a la probabilidad de que la observación o_1 haya sido generada por el estado i:

$$\delta_1(i) = \pi_i b_i(o_1) \tag{2.15}$$

siendo π_i la probabilidad inicial de cada estado.

Para obtener $\delta_1(i)$ consideramos $\pi_i = 1$, $1 \leq i \leq N$ y calculamos $b_i(o_1)$ como una mezcla de Gaussianas utilizando las ecuaciones 2.5 y 2.6.

Realizando varias iteraciones obtenemos la verosimilitud del mejor camino, pero para poder recuperar después la secuencia también necesitamos guardar el índice del mejor camino. Para ello, se define la variable $\phi_t(j)$ que guarda el estado i que hace máxima la ecuación 2.14 en cada instante de tiempo t.

$$\phi_t(j) = \arg \max_{1 \leq i \leq N} \Big[\delta_t(i)(a_{ij})\Big] \tag{2.16}$$

Recorriendo esta variable hacia atrás para los instantes de tiempo $T - 1, T - 2, ..., 1$ (proceso conocido como *Backtracking*), obtenemos la secuencia óptima de estados.

2.3.2 Reestimación de los parámetros utilizando el algoritmo de Baum-Welch

Los parámetros a_{ij} y $b_j(o_t)$ de cada HMM se eligen de manera que $P(O|M)$ sea maximizada localmente usando un procedimiento iterativo como es el método de Baum-Welch.

La función de probabilidad de la secuencia de salida se representa con una mezcla de Gaussianas, tal y como se indica en las ecuaciones 2.5 y 2.6. Así, para determinar los parámetros de cada estado j de un HMM, tendremos que estimar las medias y covarianzas de estas mezclas de Gaussianas $b_j(o_t)$.

Para ello, dividimos el vector con los valores de los ángulos utilizados en el entrenamiento, entre los distintos estados del HMM, teniendo en cuenta la secuencia óptima de estados de Viterbi explicada en el apartado anterior. A continuación, calculamos $L_j(t)$ que es la probabilidad de estar en el estado j en el instante de tiempo t, con valores iniciales de la media y la covarianza. Una vez que tenemos el valor de la verosimilitud, reestimamos la media y la covarianza utilizando las siguientes ecuaciones:

$$\hat{\mu_j} = \frac{\sum_{t=1}^{T} L_j(t) o_t}{\sum_{t=1}^{T} L_j(t)} \tag{2.17}$$

$$\hat{\Sigma_j} = \frac{\sum_{t=1}^{T} L_j(t)(o_t - \mu_j)(o_t - \mu_j)^T}{\sum_{t=1}^{T} L_j(t)} \tag{2.18}$$

Este proceso se repite hasta que el valor de la verosimilitud no cambie. Para visualizar esto, representamos el valor de la log-verosimilitud en cada iteración (ecuación 2.10):

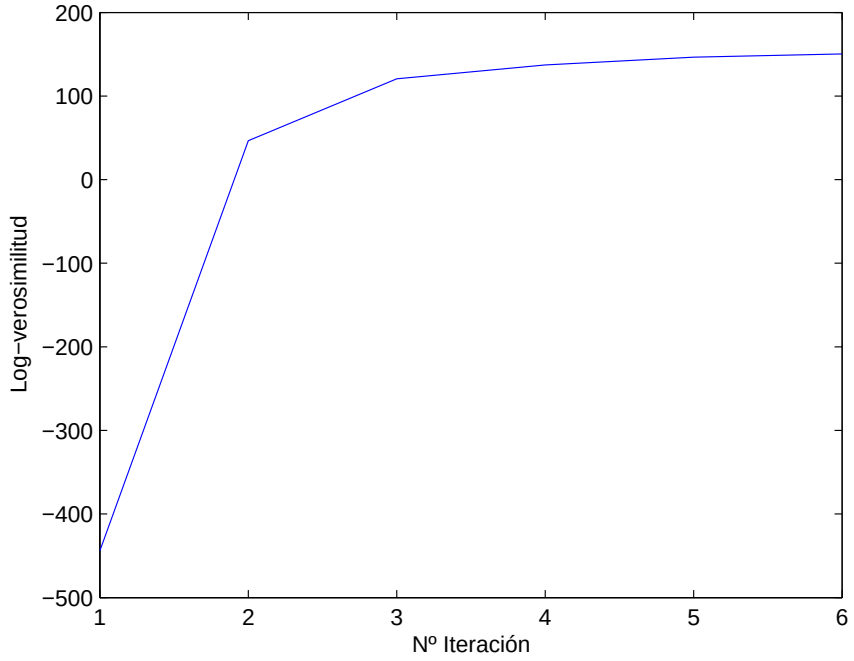


Figure 2.5: Log-verosimilitud obtenida en cada iteración (ecuación 2.10) al entrenar el modelo correspondiente al movimiento de levantar el brazo

En el ejemplo de la figura 2.5 vemos que a partir de la cuarta iteración, la verosimilitud aumenta muy poco. Por tanto, podemos dejar de iterar porque apenas se va a apreciar mejora en el resultado.

2.4 Reconocimiento de gestos por Viterbi

En el apartado anterior 2.3, se ha descrito la restimación de los parámetros del HMM utilizando el algoritmo de Baum-Welch. Una vez que hemos entrenado el modelo, vamos a reconocer el gesto realizado basándonos en aquella secuencia de estados con la que obtegamos la mayor verosimilitud (algoritmo de Viterbi). Para que el algoritmo sea más preciso, vamos a calcular

la log-verosimilitud. De esta forma, la ecuación 2.14 se convierte en,

$$\delta_{t+1}(j) = \max_{1 \leq i \leq N} \Big[\delta_t(i) + \log\left(a_{ij}\right) \Big] + \log\left(b_j(o_{t+1})\right) \tag{2.19}$$

Como muestra la figura 2.6 de [2], podemos visualizar el algoritmo de Viterbi como la búsqueda del mejor camino en una matriz donde el eje vertical representa los estados del HMM y en el eje horizontal se representa el tiempo. Cada uno de los puntos de la figura, representa la probabilidad de observar esa salida en ese instante de tiempo y cada uno de los arcos representa la probabilidad de transición entre estados. De esta forma, la log-verosimilitud de cada camino se calcula simplemente sumando el logaritmo de las probabilidades de transición y las probabilidades de salida de los puntos recorridos.

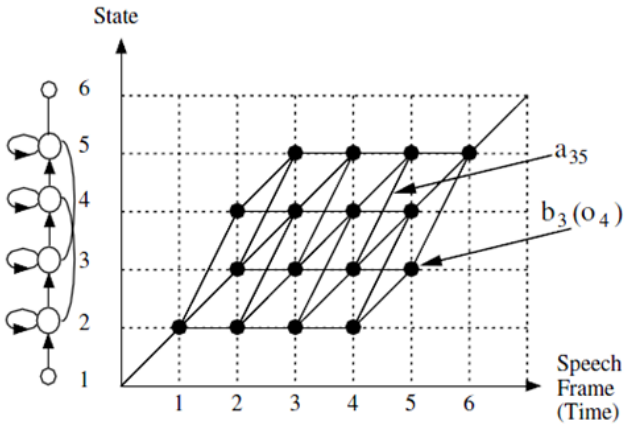


Figura 2.6: Visualización del algoritmo de Viterbi como la búsqueda del mejor camino en una matriz donde el eje vertical representa los estados del HMM y en el eje horizontal se representa el tiempo. Cada uno de los puntos, representa la probabilidad de observar esa salida en ese instante de tiempo y cada uno de los arcos representa la probabilidad de transición entre estados. Imagen extraída de [2].

En la figura 2.7 se muestra un ejemplo de esta matriz de log-verosimilitudes al realizar el movimiento de levantar el brazo. Como se indica en la barra de colores a la derecha del gráfico, el color negro muestra un valor mayor de la log-verosimilitud y el blanco el menor valor. La secuencia óptima de estados se representa con la línea azul, indicando con cada círculo, el estado en el que se encuentra en cada instante de tiempo. Observamos que el mejor camino pasa por las zonas de color más oscuro, por lo que corresponde con

la máxima log-verosimilitud.

En la figura 2.8 se muestra la matriz de log-verosimilitudes obtenida al testear el movimiento de bajar el brazo con el HMM entrenado con movimientos de subir el brazo. Comprobamos que el mejor camino no corresponde con una buena secuencia de valores de log-verosimilitud ya que las observaciones no encajan con el modelo en este caso.

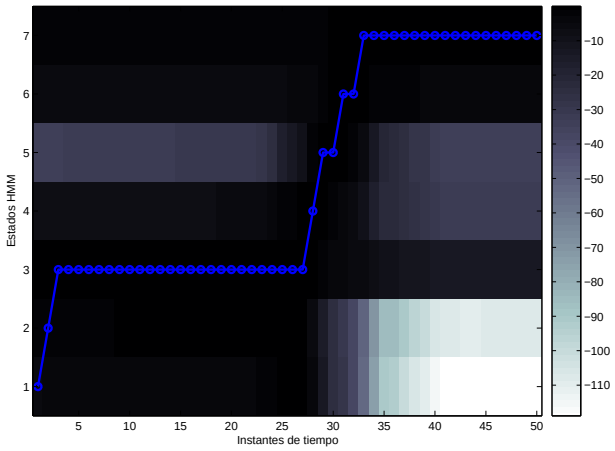


Figura 2.7: Búsqueda del algoritmo de Viterbi resultante al testear el movimiento 'levantar brazo' con su HMM

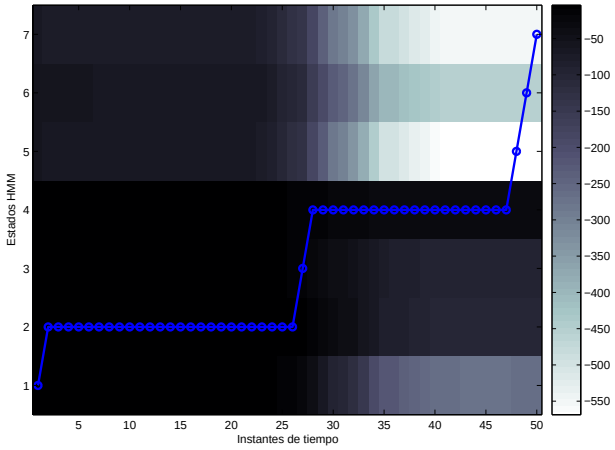


Figura 2.8: Búsqueda del algoritmo de Viterbi resultante al testear el movimiento 'bajar brazo' con un HMM 'levantar el brazo'

2.5 Análisis de prestaciones

Para analizar las prestaciones del reconocedor construido, se realizan varias pruebas de reconocimiento con Matlab y se calculan las tasas de acierto y error obtenidas con el reconocedor implementado en C++.

Se dispone de una base de datos con movimientos realizados con el brazo derecho por 4 usuarios diferentes: mover el brazo hacia arriba, hacia abajo y hacia delante, estirar y doblar el codo, hacer un círculo, una S y una Z en el aire. Tenemos 10 ficheros de cada uno de estos movimientos para realizar el entrenamiento de los modelos y 3 ficheros de test que trataremos de reconocer. Se comparan los resultados obtenidos al utilizar un HMM genérico con los obtenidos al utilizar un HMM entrenado con los movimientos del propio usuario.

2.5.1 Pruebas de reconocimiento en Matlab

Una vez construido el reconocedor en Matlab, comprobamos su correcto funcionamiento realizando varias pruebas de reconocimiento de movimientos con el brazo derecho.

La primera prueba consiste en analizar si el reconocedor es capaz de diferenciar entre los movimientos levantar y bajar el brazo. Para ello, utilizamos el ángulo que forma el cuerpo con el brazo derecho (ángulo 4 de la figura 2.4). Se obtienen los 2 HMMs con los 10 ficheros de cada movimiento y se calcula la verosimilitud con el fichero de test correspondiente a levantar el brazo. Comprobamos que con el HMM entrenado al mover el brazo hacia arriba se obtiene una verosimilitud mucho mayor que con el HMM entrenado al mover el brazo hacia abajo. Por tanto, reconoce correctamente el movimiento seleccionando áquel con el que se obtiene verosimilitud máxima.

La segunda prueba trata de diferenciar entre los movimientos mover el brazo hacia delante y doblar el codo. Primero, entrenamos los modelos utilizando solo el ángulo 4 de la figura 2.4. Calculamos la versimilitud con el fichero de test correspondiente a doblar el codo y vemos que con el HMM entrenado al mover el brazo hacia delante obtenemos mayor verosimilitud. Por tanto, se comete un error en el reconocimiento. Para evitarlo, vamos a añadir el ángulo del codo (ángulo 3 de la figura 2.4). Volvemos a entrenar los modelos y al calcular la verosimilitud comprobamos que es mayor con el HMM doblar el codo. Por tanto, añadiendo este ángulo, se reconoce correctamente el movimiento.

En la siguiente prueba, se calcula el HMM de los 7 movimientos entre-

nando los modelos con los 8 ángulos de la figura 2.4. Inicializamos el HMM con funciones de densidad de probabilidad Gaussianas de 8 dimensiones ya que tenemos 8 ángulos de entrada. A partir de los valores iniciales del HMM, se reestiman sus parámetros recorriendo los 10 ficheros de cada movimiento que tenemos en la base de datos. Se repite este proceso 6 veces, ya que observamos que no mejora la log-verosimilitud a partir de ahí, como se puede ver en la figura 2.5. Al finalizar, guardamos los parámetros de cada HMM.

Una vez obtenidos los Modelos Ocultos de Markov, vamos a realizar varias pruebas de reconocimiento con ficheros de test de cada uno de los movimientos. Para ello, se carga uno de los ficheros y se calcula la verosimilitud utilizando el algoritmo de Viterbi con los 7 HMMs que hemos entrenado. La verosimilitud obtenida con el HMM correspondiente al movimiento testeado es mayor en todos los casos. Por tanto, con este juego de datos de 7 movimientos, el reconocedor construido funciona correctamente.

Se disponde también de ficheros de test donde se han grabado los mismos movimientos pero ejecutándolos de forma diferente. En uno de ellos, se para a mitad del movimiento unos segundos y en el otro, se realiza el movimiento de forma intermitente. Se calcula la verosimilitud con los 7 HMMs y comprobamos que en todos los casos se reconoce correctamente el movimiento testeado.

2.5.2 Evaluación del reconocedor C++

Una vez analizado el correcto funcionamiento del reconocedor en Matlab, vamos a calcular las tasas de acierto y error que proporciona el reconocedor construido en C++.

Como hemos dicho anteriormente, nuestra base de datos contiene movimientos realizados por 4 usuarios diferentes. En primer lugar, se calculan los HMM de cada movimiento con los 10 ficheros disponibles por cada usuario. Para evaluar el reconocedor, no podemos utilizar ficheros de test utilizados en el entrenamiento de los HMMs ya que se producirían resultados erróneamente favorables. Por esto, se calculan 10 HMMs por usuario y movimiento, dejando fuera en cada entrenamiento uno de los ficheros que posteriormente utilizaremos para testear.

Creamos un HMM genérico a partir de los movimientos del usuario 1 para comparar la tasa de error con el HMM creado a partir de los movimientos del propio usuario. La tasa de error en el reconocimiento de movimientos

MER se define como sigue:

$$MER = \frac{Movimientos\ erroneamente\ reconocidos}{Movimientos\ totales} \tag{2.20}$$

En la tabla 2.1 se muestran los resultados obtenidos en el análisis:

Usuario	**MER genérica**	**MER**	**% reducción**
Usuario 2	37,88 %	6,06 %	84 %
Usuario 3	45,45 %	12,12 %	73,33 %
Usuario 4	40,91 %	10,60 %	74,09 %

Cuadro 2.1: MER

Como vemos en la tabla 2.1, el reconocedor funciona mejor cuando se utiliza un HMM entrenado con los movimientos del propio usuario, obteniendose mejoras superiores al 70 %. En este caso, la MER de nuestro reconocedor es bastante baja, consiguiendo una tasa de acierto media del 90,4 %. Estos resultados son interesantes ya que en la aplicación es posible entrenar los modelos para usuarios concretos.

Es cierto que estos resultados se obtienen con un número finito de movimientos, solo 6 diferentes. No obstante para cada usuario se intentan reconocer 10 ficheros de cada uno de los movimientos con 2 HMMs, uno genérico y otro creado a partir de los movimientos del propio usuario. Esto supone un total de 360 pruebas de reconocimiento, por lo que podemos considerar que la MER media obtenida representa al reconocedor construido de forma apropiada.

Capítulo 3

Descripción de la aplicación

La aplicación diseñada en este proyecto podrá ayudar a niños con movilidad reducida y problemas de interacción, teniendo como objetivo la comunicación y la terapia.

Esta aplicación consiste en distintos juegos de identificación de movimientos cuya base es el reconocedor de gestos construido. Los juegos tienen imágenes y sonidos que estimulan a los niños y consiguen captar su atención. Además pueden ayudarles a mejorar su movilidad y su capacidad de interacción con el medio que les rodea.

3.1 Estructura de la aplicación

La estructura de la aplicación se muestra en la figura 3.1:

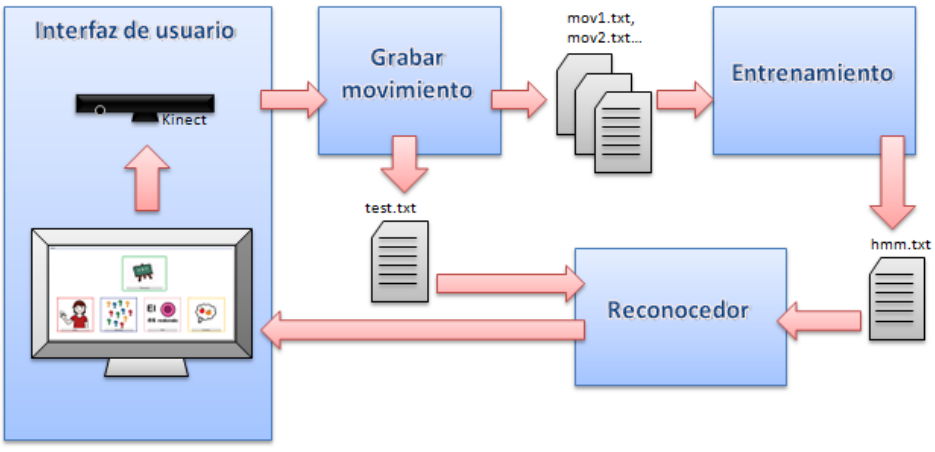


Figura 3.1: Estructura de la aplicación implementada en este proyecto

En primer lugar, es necesario entrenar el HMM correspondiente al mo-

vimiento con el que el terapeuta quiere trabajar. La aplicación dispone de un apartado para ello. Dentro de este, hay 2 opciones, una para grabar movimientos y otra para realizar el entrenamiento. Si ya tenemos varias grabaciones del movimiento, podemos entrenar el modelo directamente. En caso contrario, necesitamos grabar varias veces el gesto, seleccionándo la opción *Grabar movimiento*. Para crear un HMM más preciso, podemos utilizar grabaciones del movimiento realizadas por los niños con los que vamos trabajar y añadirlas a nuestro juego de datos para entrenar el modelo. La cámara del Kinect comenzará a grabar cuando detecte un esqueleto y guardará los datos en ficheros de texto, asignándoles nombre de forma secuencial *mov1.txt, mov2.txt, etc*. Una vez que tenemos ficheros con los datos de los movimientos, realizamos el entrenamiento para crear el HMM correspondiente, seleccionando la opción *Training*. El HMM resultante, se guarda en el fichero de texto *hmm.txt*.

Una vez calculados los HMM de los movimientos que vamos a utilizar, el terapeuta puede seleccionar uno de los cuatro juegos. El niño realizará el movimiento que se indique en el juego, ya sea a través de imágenes o adivinanzas, y se guardará en el fichero de texto *test.txt*. El reconocedor construido comparará dicho fichero con el HMM correspondiente y calculará la verosimilitud. En función de ella, determinará si el gesto se ha realizado correctamente o no.

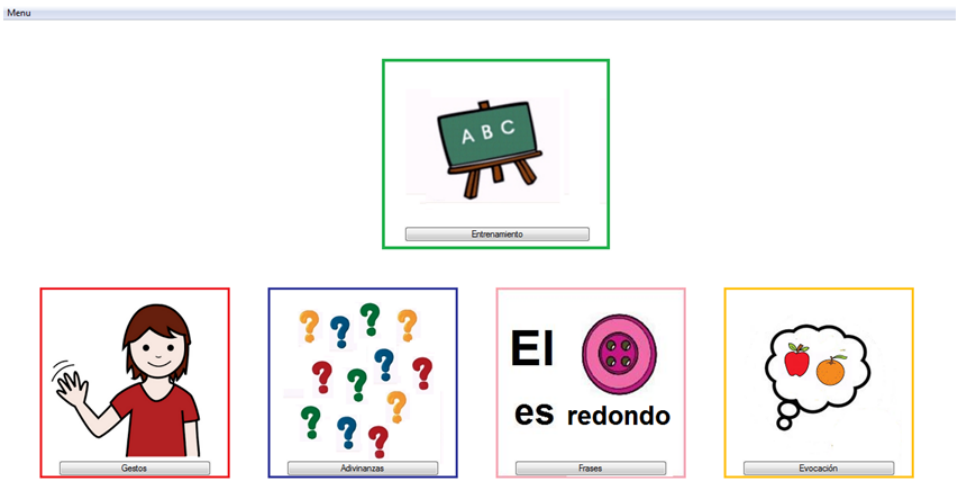


Figura 3.2: Pantalla de inicio de la aplicación: Menú de configuración, Entrenamiento de los modelos y 4 juegos: Gestos, Adivinanzas, Frases y Evocación

La pantalla principal de la aplicación se muestra en la figura 3.2. Dispone de un menú de configuración, un apartado para entrenar los distintos movimientos y 4 juegos: Gestos, Adivinanzas, Frases y Evocación. Los pic-

togramas utilizados se han obtenido del portal ARASAAC[1] que ofrece recursos gráficos y materiales para la comunicación alternativa y aumentativa.

En los siguientes apartados se explican los distintos bloques que componen la aplicación (figuras 3.1 y 3.2).

3.2 Edición y configuración

El terapeuta dispone de un Menú de configuración genérico y uno individual para cada juego.

En el Menú de la pantalla principal se debe seleccionar el directorio de trabajo (figura 3.3). Aquí se guardarán todos los ficheros e imágenes utilizados en los distintos juegos.

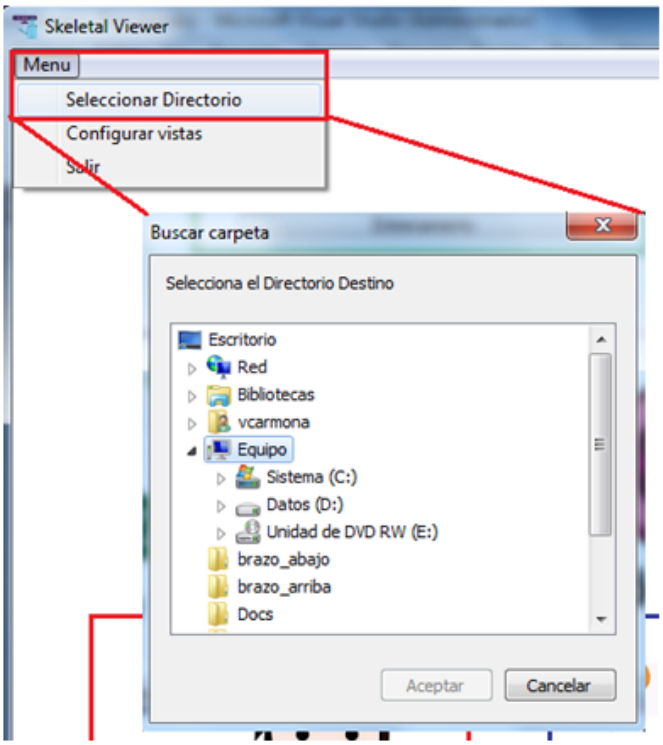


Figura 3.3: Selección del directorio de trabajo en el menú de la pantalla principal

Otra configuración común a todos los juegos es seleccionar si la grabación

[1] www.arasaac.org

del vídeo y la vista del esqueleto (figura 3.4) deben estar visibles o no. Se marcará la vista que queramos visualizar en la pantalla principal de cada juego o se dejarán desmarcadas en caso de desear que queden ocultas. (figura 3.5)

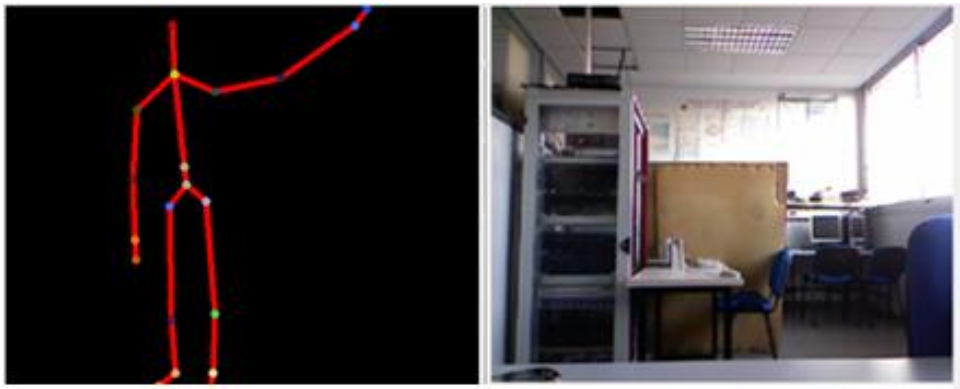


Figura 3.4: Vistas de la grabación del vídeo y del esqueleto generadas por Kinect

Figura 3.5: Configuración de las vistas de grabación del vídeo y del esqueleto para que aparezcan o no en la pantalla de cada juego

La configuración de cada juego (figura 3.6) consiste en seleccionar el gesto a realizar por el niño y el nivel de dificultad (Alta, Media o Baja). La lista de gestos se completará con todos los movimientos con fichero HMM en el directorio de trabajo. Cuanto más alto sea el nivel de dificultad, mayor será el umbral de verosimilitud para reconocer el gesto realizado. Por tanto, será necesario realizar el gesto de forma más precisa para que reconozca el movimiento correctamente.

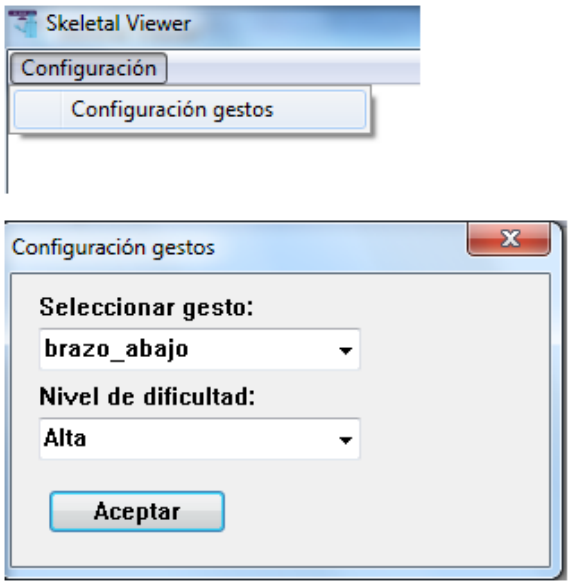


Figura 3.6: Configuración de los juegos: Selección del gesto y nivel de dificultad

3.3 Entrenamiento

Para comenzar a utilizar la aplicación necesitamos disponer de los HMMs de aquellos movimientos que queramos realizar en los juegos. El apartado de Entrenamiento nos permite obtener los modelos ocultos de Markov de cualquier movimiento. En primer lugar, tenemos que grabar varias repeticiones del mismo gesto pulsando el botón *Grabar movimiento*' y a continuación, realizar el entrenamiento pulsando '*Training*' (figura 3.7).

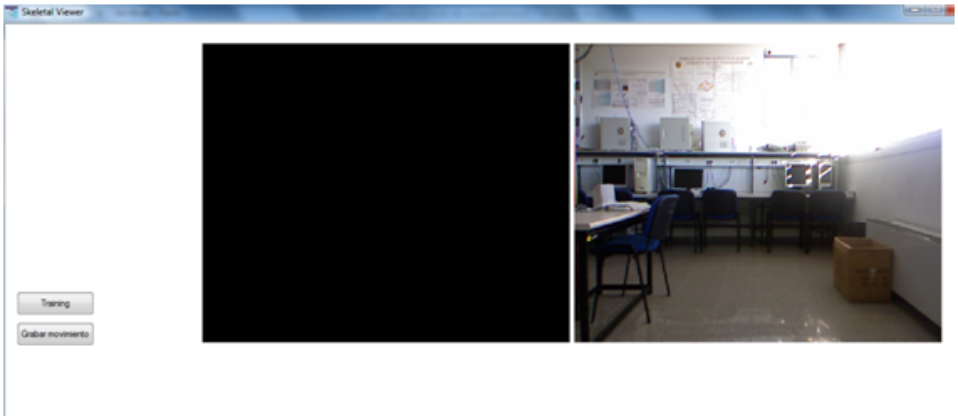


Figura 3.7: Pantalla principal del apartado Entrenamiento. Permite grabar los movimientos y obtener sus HMMs (Training)

Para implementar este reconocedor de movimientos en lenguaje C++, que es el lenguaje de programación usado en el SDK de Kinect, lo primero que necesitamos son los tipos matriz, Gaussiana y mezcla de Gaussianas. Creamos también funciones básicas para el procesado de matrices en C++ (Apéndice A.1). Para crear la interfaz de usuario, utilizamos las funciones API de Windows (Apéndice A.2). La clase CSkeletalViewerApp del SDK[2] nos permite acceder y manipular los datos capturados por el sensor Kinect, así como mostrar las imágenes del video y del esqueleto (Apéndice A.3). Todo lo relacionado con la implementación de la aplicación en lenguaje C++ se detalla en el apéndice A.

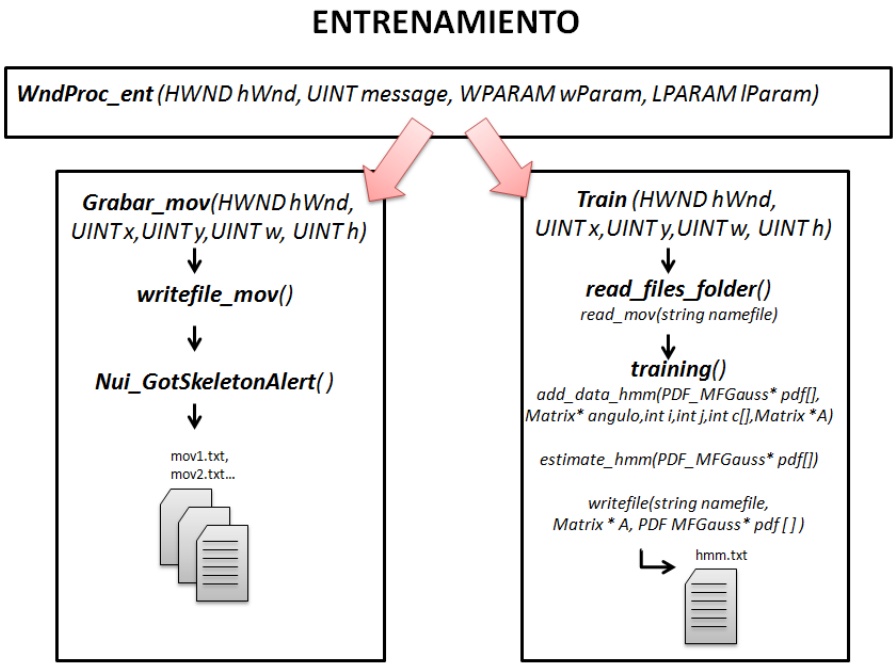


Figura 3.8: Esquema con las funciones implementadas para grabar los movimientos y entrenar los HMMs

El procedimiento WndProc_ent recibe los mensajes generados en la ventana entrenamiento. Al inicializar la ventana (mensaje WM_INITDIALOG) se crean los botones '*Grabar movimiento*' y '*Training*'. Para procesar los mensajes que recibe cada botón, utilizamos la técnica de subclasificación explicada en el ápendice A.2. Para ello, implementamos las siguientes funciones:

```
Grabar_mov (HWND hWnd, UINT x,UINT y,UINT w, UINT h)
Train (HWND hWnd, UINT x,UINT y,UINT w, UINT h)
```

[2] `https://dev.windows.com/en-us/kinect`

Grabar movimiento: Al pulsar este botón (mensaje WM_LBUTTONUP), se llama a la función writefile_mov() que da nombre a los ficheros de texto donde se van a guardar los datos del movimiento grabado, *mov1.txt, mov2.txt, etc*. Estos ficheros se guardarán en el directorio seleccionado en la pantalla principal de la aplicación. En caso de que el usuario no haya seleccionado ningún directorio, aparecerá el mensaje de la figura 3.9.

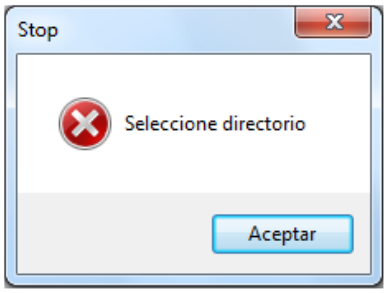


Figura 3.9: Mensaje de error: No se ha seleccionado ningún directorio de trabajo

Desde esta función se activará la grabación del movimiento. Para ello, se utiliza la clase CSkeletalViewerApp del SDK de Kinect. Cuando la cámara detecta el esqueleto, el método Nui_GotSkeletonAlert( ) captura las coordenadas de los 20 puntos de la figura 2.2 durante 2,5 segundos. Puesto que la tasa de captura de Kinect es de $20fps$, se obtienen 50 capturas con la posición de los 20 puntos del esqueleto. Estas coordenadas se guardan en una matriz de dimensiones 50x60 (50 filas, una por cada captura de posiciones y 60 columnas - 20 puntos del esqueleto x 3 coordenadas). A continuación, se crean los ficheros de texto con esta información.

Training: Al pulsar este botón (mensaje WM_LBUTTONUP), se llama a la función read_files_folder () que lee los ficheros *mov1.txt, mov2.txt, etc* del directorio seleccionado en la pantalla principal de la aplicación. Estos ficheros contienen las grabaciones de los movimientos con los que realizar el entrenamiento y construir el HMM.

Para leer los ficheros de texto con las posiciones de los puntos del esqueleto, creamos la siguiente función,

```
Matrix* read_mov(string namefile)
```

Después de leer las coordenadas del movimiento capturado con la Kinect, calculamos el valor de los ángulos en todos los instantes de tiempo, tal y como se comenta en el apartado 2.2 de esta memoria. Cada fila de la matriz que devuelve la función read_mov, contendrá los valores de uno de los ángulos. Como ya hemos dicho, en el desarrollo de este proyecto se van a

utilizar los 8 ángulos mostrados en la figura 2.4.

A continuación, se llama a la función training(). Si en el directorio no hay ningún fichero *mov1.txt, mov2.txt, etc* salta el mensaje de la figura 3.10. En caso contrario, se realiza el entrenamiento con los ficheros que haya en el directorio.

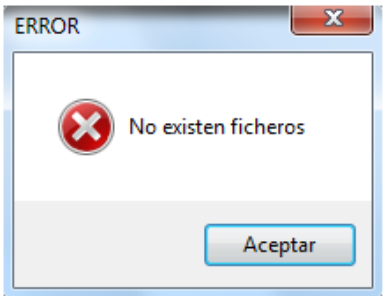


Figura 3.10: Mensaje de error: En el directorio de trabajo no existen ficheros para realizar el entrenamiento

Lo primero que tenemos que realizar para entrenar el HMM, es inicializar sus parámetros. Construimos modelos con matriz de transición de tipo left-to-right 2.1. Las funciones de probabilidad de salida de cada estado son una mezcla de Gaussianas. Para inicializar la media utilizamos la función rand_float() y la covarianza será la matriz identidad de dimensión D. D es la dimensión de los datos de entrada al reconocedor, es decir, el número de ángulos obtenidos con la función read_mov.

Una vez inicializados los parámetros del HMM, utilizamos la siguiente función para calcular la secuencia óptima de Viterbi y con ella, asignar los valores de los ángulos de cada fichero, a los distintos estados del HMM.

```
void add_data_hmm(PDF_MFGauss* pdf[], Matrix* angulo, int i, int j
    , int c[], Matrix *A)
```

- pdf[] es la mezcla de Gaussianas del HMM.
- angulo es la matriz que devuelve read_mov al leer cada uno de los ficheros de texto utilizados en el entrenamiento.
- i indica el número de iteración.
- j índice del fichero de texto.
- c[] es el número de columna de cada estado del HMM donde añadir los valores de los ángulos.
- A es la matriz de transición del HMM.

Una vez recorridos todos los ficheros utilizados en el entrenamiento, se llama a la función que estima los parámetros del HMM,

```
void estimate_hmm(PDF_MFGauss* pdf[])
```

De esta forma, en cada iteración calculamos de nuevo los parámetros de la mezcla de Gaussianas del HMM (pdf[]) obteniendo valores cada vez más precisos. Repetimos este proceso durante 6 iteraciones. Como se ha visto en el apartado 2.3.2 de esta memoria, no hace falta realizar más iteraciones porque apenas se va apreciar mejora en el resultado.

Una vez realizado el entrenamiento, creamos la función `write_file` para guardar los parámetros del HMM.

```
void write_file(string namefile, Matrix* A, PDF_MFGauss* pdf[])
```

Con ella, grabamos el modelo resultante en el fichero de texto *hmm.txt*. En este fichero, se guarda el número de estados y de componentes del HMM, así como su matriz de transición. Para cada estado del HMM guardamos los pesos, la media y la matriz de covarianza de cada una de las Gaussianas de la mezcla. Cuando el HMM se ha creado correctamente, aparece el mensaje de la figura 3.11 para informar al usuario de que ya puede utilizar este movimiento en los distintos juegos.

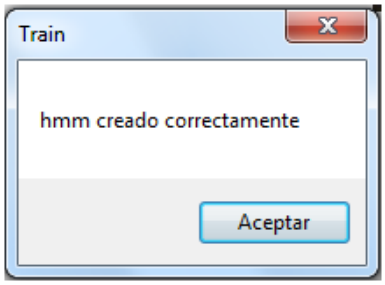


Figura 3.11: Mensaje informativo: El entrenamiento ha terminado con éxito y el HMM se ha creado correctamente

3.4 Juegos

Como podemos ver en la figura 3.2, la aplicación tiene 4 juegos diferentes: Gestos, Adivinanzas, Frases y Evocación. Mediante estos juegos se pretende mejorar la capacidad motora y coordinación de niños con movilidad reducida, trabajando diferentes movimientos.

El grupo Vivolab desarrolló la herramienta "Vocaliza"[3], una aplicación de asistencia a la logopedia para mejorar la capacidad del habla en niños con

[3]Proyecto Comunica `http://dihana.cps.unizar.es/~alborada/herramientas.html`

alteraciones en el lenguaje [6]. La aplicación desarrollada en este proyecto es muy similar ya que servirá de ayuda a la terapia de niños y personas con necesidades especiales.

Los juegos tienen opciones de configuración para que el terapeuta pueda seleccionar los distintos movimientos, las imágenes y el nivel de dificultad. El funcionamiento de todos los juegos es similar: se muestra una imagen y/o un texto que los niños tendrán que asociar a un movimiento y reproducirlo. La cámara Kinect grabará dicho movimiento y lo comparará con el HMM correspondiente. Según el nivel de dificultad seleccionado, el umbral de verosimilitud para reconocer el gesto será más o menos alto. Se mostrará una imagen y se reproducirá un sonido para indicar al niño si ha realizado correctamente el gesto o debe intentarlo de nuevo.

Reconocedor: Este bloque compara el movimiento realizado por el niño (*test.txt*) con el HMM seleccionado por el terapeuta (*hmm.txt*).

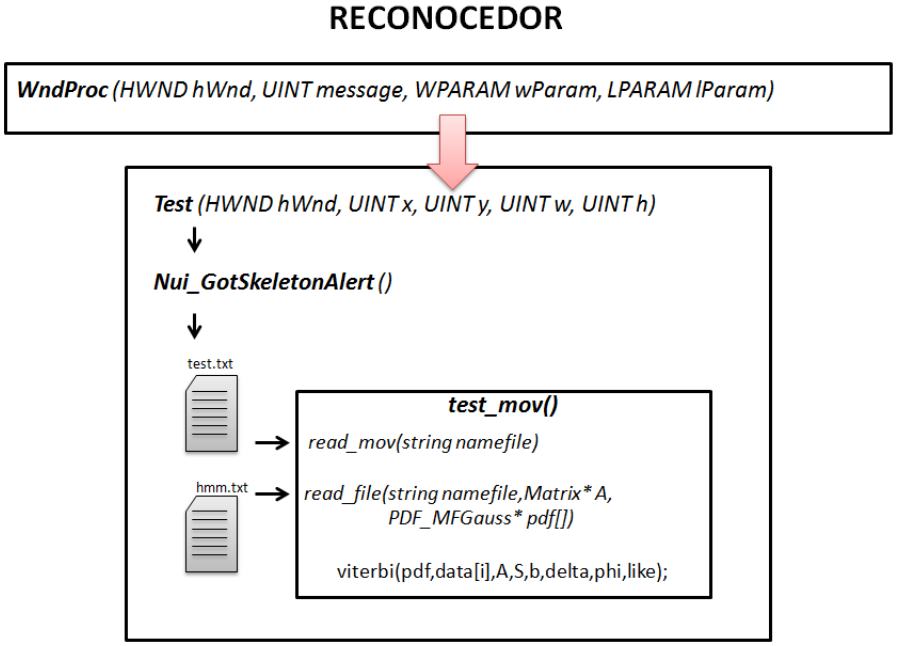


Figura 3.12: Esquema con las funciones implementadas para reconocer el gesto realizado

El procedimiento WndProc recibe los mensajes generados en la ventana de cada juego. Al inicializar la ventana (mensaje WM_INITDIALOG) se crea el botón '*Test*'. Para procesar los mensajes que recibe este botón, utilizamos la técnica de subclasificación explicada en el ápendice A.2. Para ello,

implementamos las siguiente función:

```
Test (HWND hWnd, UINT x,UINT y,UINT w, UINT h)
```

Al pulsar el botón '*Test*' (mensaje WM_LBUTTONUP), se activa la grabación del movimiento. Para ello, se utiliza la clase CSkeletalViewerApp del SDK de Kinect. Cuando la cámara detecta el esqueleto, el método Nui_GotSkeletonAlert( ) captura las coordenadas de los 20 puntos de la figura 2.2 y crea el fichero de texto *test.txt*. A continuación, se llama a la función test_mov().

Tal y como hemos hecho antes para leer los ficheros en el entrenamiento, utilizamos la función Matrix* read_mov(string namefile) para leer el fichero *test.txt* con el movimiento realizado por el niño. Esta función devuelve la matriz angulo que contiene los ángulos de dicho movimiento.

Para leer el fichero con el HMM seleccionado por el terapeuta (namefile), utilizamos la función read_file que nos devuelve la matriz de transición A y para cada estado del HMM, los pesos, la media y la matriz de covarianza de cada una de las Gaussianas de la mezcla pdf[],

```
void read_file(string namefile,Matrix* A,PDF_MFGauss* pdf[])
```

A continuación, calculamos la verosimilitud obtenida con el movimiento realizado por el niño, utilizando el algoritmo de Viterbi,

```
void viterbi (PDF_MFGauss* pdf[], Matrix* angulo, Matrix* A,
    Matrix *S, Matrix *b, Matrix *delta, Matrix *phi,float *like)
```

- pdf[] es la mezcla de Gaussianas del HMM.
- angulo es la matriz que devuelve read_mov al leer el fichero *test.txt*.
- A es la matriz de transición del HMM.
- S es la secuencia óptima de estados.
- b es la función de probabilidad de la secuencia de salida, $b_j(o_t)$ ecuación 2.5.
- delta es la probabilidad del mejor camino que calculamos con la ecuación 2.19, $\delta_{t+1}(j)$
- phi es la secuencia de estados que maximiza la probabilidad, $\phi_t(j)$ ecuación 2.16
- like es la log-verosimilitud obtenida.

El último paso es comparar la log-verosimilitud con el umbral (este valor depende del nivel de dificultad seleccionado por el terapeuta). En caso de que la log-verosimilitud obtenida sea superior a dicho umbral, se reconoce

el gesto realizado por el niño, en caso contrario deberá intentarlo de nuevo.

En los siguientes subapartados se describen los 4 juegos de la aplicación.

3.4.1 Gestos

Con el juego 'Gestos', los niños podrán practicar la realización de distintos movimientos, por lo que les ayudará a mejorar su capacidad motora.

En el directorio seleccionado en la pantalla principal (figura 3.3), tendremos los ficheros con los HMM de los movimientos que hayan sido entrenados. El nombre de estos ficheros debe comenzar por *hmm_* y a continuación indicar el nombre del movimiento correspondiente. De esta forma, en el desplegable de las opciones de configuración, aparecerán todos los movimientos que tengan HMM en el directorio. Para configurar este juego, el terapeuta debe seleccionar uno de los movimientos de este desplegable y el nivel de dificultad (Alta, Media, Baja), tal y como se ha explicado en el apartado 3.2 de esta memoria.

Una imagen asociada al movimiento seleccionado aparacerá en la pantalla si existe en el directorio de trabajo. Deberá ser un archivo *.bmp* cuyo nombre debe comenzar por *img_* y a continuación el nombre del movimiento correspondiente. En la figura 3.14 se muestra la pantalla principal del juego Gestos donde el terapeuta ya ha seleccionado un movimiento que tiene asociada la imagen de la niña saludando.

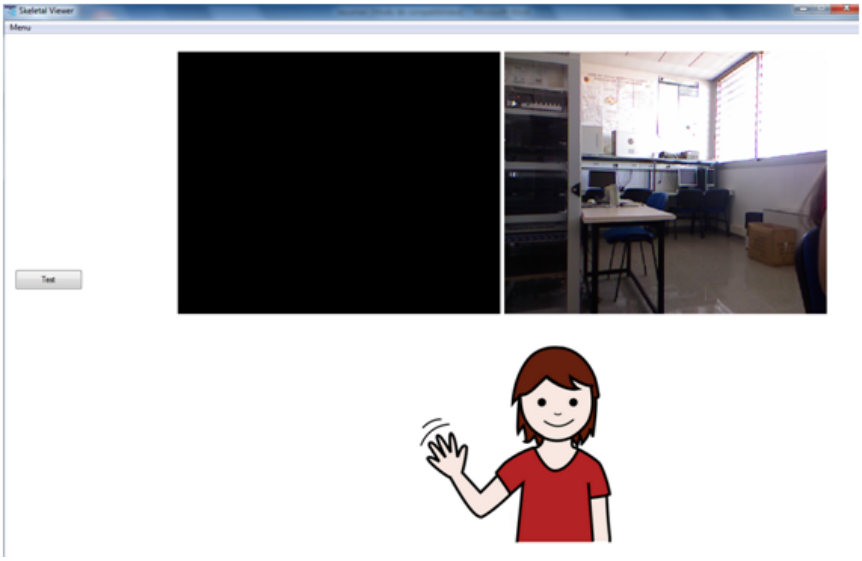


Figura 3.13: Pantalla del Juego Gestos una vez que el terapeuta ha seleccionado el movimiento con la imagen asociada "niña saludando".

Cuando el niño esté preparado, el terapeuta pulsa el botón 'Test' y en el momento en que se detecte un esqueleto delante de la cámara Kinect, se grabará el movimiento que realice el niño en un fichero de texto (*test.txt*). Este fichero y el fichero con el HMM seleccionado serán los datos de entrada al reconocedor con el que se calcula la verosimilitud del movimiento realizado por el niño.

Si la verosimilitud obtenida es superior al umbral (este valor depende de la dificultad seleccionada por el terapeuta), se pintará un tick verde en la pantalla 3.14a y se reproducirá un aplauso porque el movimiento se habrá realizado correctamente. En caso contrario, se pintará un aspa roja 3.14b para indicar al niño que debe intentarlo de nuevo.

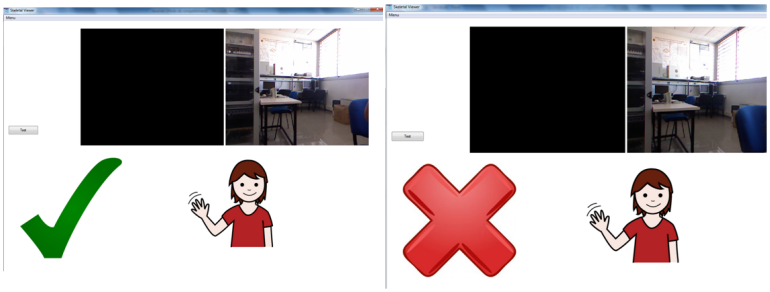


Figura 3.14: Resultado del juego Gestos. En 3.14a el movimiento ha sido realizado correctamente (verosimilitud superior al umbral). En 3.14b el movimiento no se ha realizado correctamente (verosimilitud por debajo del umbral).

3.4.2 Adivinanzas

El juego 'Adivinanzas', permite que los niños razonen y asocien imágenes con sus gestos, además de poder practicar la realización de movimientos. Por tanto, este juego servirá al terapeuta como herramienta de ayuda para mejorar la capacidad de razonamiento de los niños, así como su capacidad motora.

La configuración de este juego es igual que en el juego Gestos. El terapeuta debe seleccionar el movimiento con el que quiere trabajar y el nivel de dificultad (figura 3.6). Una vez seleccionado el movimiento, se mostrará la adivinanza correpondiente que estará escrita en un fichero de texto en el directorio de trabajo. Para relacionar las adivinanzas con los movimientos, el nombre del fichero debe comenzar por *adiv_* y a continuación el nombre del movimiento. Si no existe ningún fichero correspondiente al movimiento seleccionado, aparece el mensaje de error de la fichero 3.15.

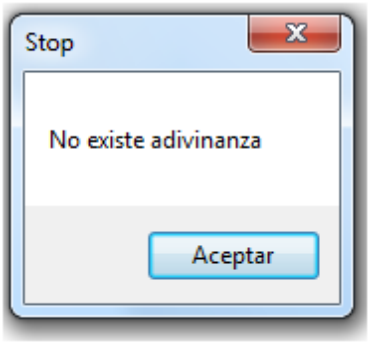


Figura 3.15: Mensaje de error: En el directorio de trabajo no existe una adivinanza asociada al movimiento seleccionado por el terapeuta

En la figura 3.16, se muestra un ejemplo de adivinanza. Se ha seleccionado el movimiento Z y en el directorio existe el fichero *adiv_Z.txt* que contiene la frase 'Última letra del abecedario'.

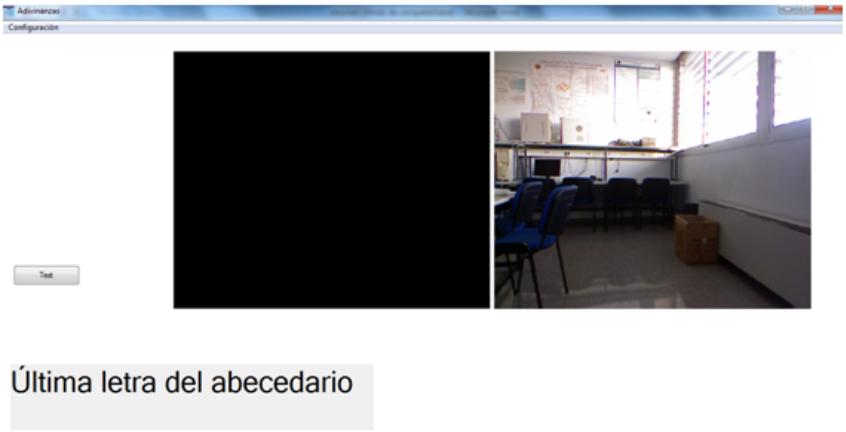


Figura 3.16: Pantalla del Juego Adivinanzas una vez que el terapeuta ha seleccionado el movimiento Z. En el directorio de trabajo, existe el fichero *adiv_Z.txt* que contiene la frase 'Última letra del abecedario'.

Cuando el niño averigüe el gesto que tiene que realizar para resolver la adivinanza, el terapeuta pulsa el botón 'Test' y en el momento en que se detecte un esqueleto delante de la cámara Kinect, se grabará el movimiento que realice el niño en un fichero de texto (*test.txt*). A continuación, el reconocedor construido testea dicho movimiento con el HMM correspondiente y calcula la verosimilitud.

Si el niño acierta la adivinanza, la verosimilitud obtenida será mayor que el umbral. En este caso, se muestra la imagen asociada al movimiento

para indicar al niño que ha acertado (figura 3.17). Esta imagen debe ser un archivo *.bmp* de nombre *img_* y a continuación el nombre del movimiento correspondiente. Si no existe ninguna imagen en el directorio de trabajo con esta nomenclatura, se mostrará un tick verde en caso de que el niño haya adivinado el movimiento. En caso de no acertar la adivinanza, aparecerá un aspa roja. Podemos visualizar esto en el ejemplo de adivinanza de la figura 3.18. Se ha seleccionado el movimiento S y en el directorio existe el fichero *adiv_S.txt* que contiene la frase 'Letra para formar el plural'. Sin embargo, no existe el fichero *img_S.bmp* con la imagen de la letra S.

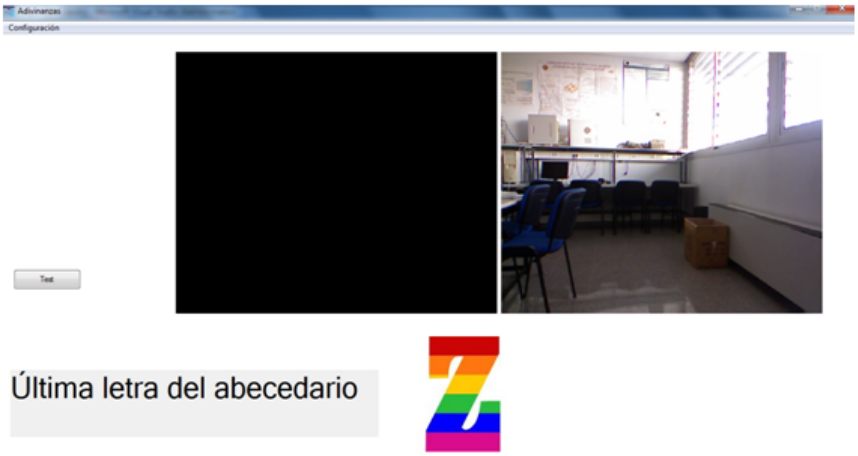


Figura 3.17: Resultado del juego Adivinanzas al realizar una Z en el aire (verosimilitud superior al umbral). En el directorio de trabajo existe el fichero *img_Z.bmp* que contiene la imagen de la letra Z

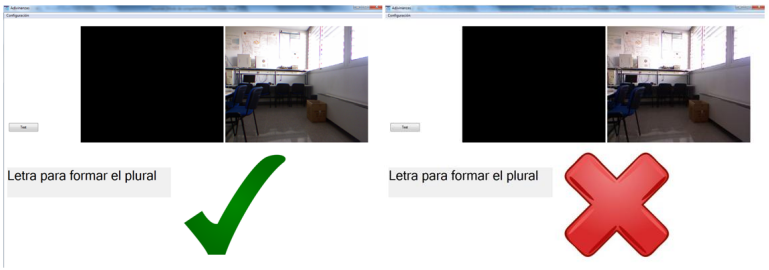


Figura 3.18: Resultado del juego Adivinanzas. En 3.18a se ha realizado una S en el aire (verosimilitud superior al umbral), pero no existe el fichero *img_S.bmp* con la imagen de la letra S. En 3.18b se ha realizado otro movimiento diferente a la letra S (verosimilitud por debajo del umbral)

3.4.3 Frases

Con el juego 'Frases', los niños podrán entender la formación de oraciones, así como asociar las imágenes a los movimientos. Este juego mejora tanto su capacidad de comprensión como su capacidad motora.

Este juego se configura de la misma forma que los juegos anteriores. El terapeuta debe seleccionar un movimiento y el nivel de dificultad (figura 3.6). Para mostrar la oración correspondiente al movimiento seleccionado, es necesario guardar una imagen en formato *.bmp* en el directorio de trabajo. El nombre de esta imagen debe comenzar por *frase_* y a continuación el nombre del movimiento correspondiente. Si no existe ningún fichero con estas características, aparece un mensaje informativo para que el terapeuta seleccione otro movimiento o guarde la imagen correspondiente en el directorio de trabajo. Al representar la frase de texto en pantalla se sustituirá la palabra por su imagen, como se muestra en la figura 3.19.

Como ejemplo, creamos la imagen *frase_circulo.bmp*. Seleccionamos el movimiento círculo y en la pantalla del juego aparece la frase que vemos en la figura 3.19.

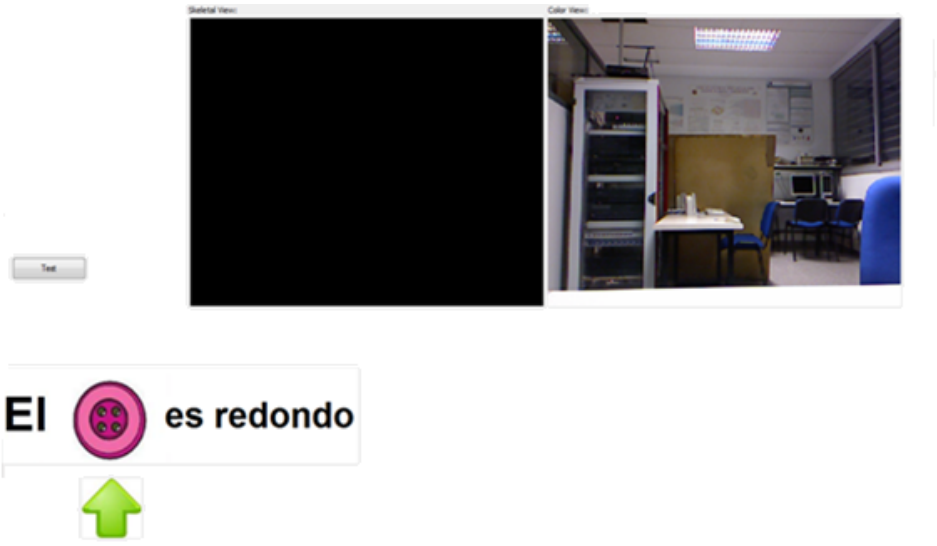


Figura 3.19: Pantalla del Juego Frases una vez que el terapeuta ha seleccionado el movimiento círculo. En el directorio de trabajo, existe el fichero *frase_circulo.bmp* que contiene la frase seleccionada para el juego

La imagen sustituye a una palabra que el niño debe deducir para forma la oración y asociarla con un movimiento que debe reproducir. Tal y como

se ha hecho en los juegos anteriores, el movimiento realizado por el niño se guarda en un fichero de texto (*test.txt*) y el reconocedor calcula su verosimilitud con el HMM correspondiente.

Si el niño consigue realizar correctamente el movimiento que se adapta a la imagen, la verosimilitud obtenida será mayor que el umbral (este valor depende del nivel de dificultad seleccionado por el terapeuta). En este caso, la flecha verde que señala la imagen se convierte en un tick verde indicando al niño que ha acertado. En caso contrario, aparecerá un aspa roja (figura 3.20).

Figura 3.20: Resultado del juego Frases. En 3.20a se ha realizado un círculo en el aire (verosimilitud superior al umbral). En 3.20b el movimiento realizado es diferente (verosimilitud por debajo del umbral)

3.4.4 Evocación

El juego 'Evocación' desarrolla la capacidad de imaginación y la memoria del niño, ya que puede pensar en la imagen que desee y asociarla a un movimiento. Si lo reproduce de forma adecuada, se mostrará la imagen correspondiente a dicho movimiento.

Este juego no tiene menú de configuración, el movimiento realizado por el niño se testeará con todos los HMM disponibles en el directorio de trabajo. El terapeuta puede preparar con antelación directorios de trabajo con vocabularios específicos sobre las palabras y gestos que quiera trabajar. Se mostrará la imagen correspondiente a áquel movimiento con el que se obtenga mayor verosimilitud. Por tanto, el terapeuta debe guardar los HMM de diferentes movimientos *hmm_xxx.txt* y su imagen asociada *img_xxx.bmp* en el directorio de trabajo. Cuando el niño esté preparado y haya pensado qué movimiento va a realizar, el terapeuta debe pulsar el botón de *Test* para que la cámara de Kinect comience a grabar. El movimiento se guardará en un fichero de texto *test.txt* que posteriormente se comparará con los HMM disponibles.

En la figura 3.21 se muestra el resultado obtenido al realizar 3 movimientos diferentes, un círculo, una Z y una S.

Figura 3.21: Resultado del Juego Evocación. La imagen mostrada corresponde a áquel movimiento con el que se haya obtenido mayor verosimilitud: círculo, Z y S respectivamente.

Capítulo 4

Conclusiones y Líneas futuras

En este proyecto se ha implementado un reconocedor de secuencias gestuales utilizando la Kinect basado en modelos ocultos de Markov.

Se ha construido una base de datos con diferentes movimientos realizados con el brazo derecho. Con la cámara Kinect se ha grabado la posición (x, y, z) de 20 puntos del esqueleto en distintos instantes de tiempo. Hemos representado estas coordenadas en Matlab generando un video que muestra dichos movimientos. Utilizando estos ficheros, se calcula el valor de 8 ángulos del cuerpo que serán los datos de entrada al reconocedor construido. Hemos comprobado que se trata de una característica de los movimientos significativa para su identificación ya que podemos determinar de qué movimiento se trata según el valor de los ángulos en las distintas capturas realizadas. Una mejora para futuras aplicaciones, sería dar versatilidad a la hora de calcular los ángulos, de forma que el usuario pueda elegir áquellos que considere relevantes para los movimientos de estudio.

Otro objetivo del proyecto es realizar el entrenamiento iterativo de los movimientos para crear modelos ocultos de Markov que nos sirven para el reconocimiento utilizando el algoritmo de Viterbi. En primer lugar, se ha implementado el sistema con Matlab analizando su correcto funcionamiento al realizar varias pruebas con los ficheros disponibles en la base de datos. Se ha demostrado que también es posible reconocer movimientos realizados a distinta velocidad como parando a mitad de la ejecución o realizando el movimiento de forma intermitente.

En segundo lugar, se ha desarrollado el sistema en lenguaje C++ que es el lenguaje de programación utilizado en el SDK de Kinect. Para calcular las tasas de acierto y error, hemos utilizado movimientos implementados por

4 usuarios diferentes consiguiendo resultados satisfactorios en todos los casos. Se han comparado los movimientos reconocidos correctamente cuando se utiliza un HMM genérico y un HMM creado a partir de los movimientos del propio usuario, consiguiendo una tasa de acierto en el reconocimiento mayor en el último caso.

Por último, se ha implementado una aplicación basada en el reconocimiento de gestos. Esta aplicación ha conseguido cumplir los objetivos planteados inicialmente, ya que resulta una herramienta útil para los terapeutas que ayudan a niños a mejorar su capacidad de movimiento. Se han utilizado imágenes sencillas y coloridas que captan la atención de los niños con problemas de interacción con el medio que les rodea. Por otra parte, dispone de varias opciones de configuración que permite al terapeuta trabajar con usuarios con distintas capacidades de movimiento, así como entrenar sus propios modelos para ser más preciso en el reconocimiento. Además, a través de los juegos desarrollados en la aplicación, los niños podrán practicar diferentes movimientos y relacionarlos con imágenes, mejorando así su capacidad de razonamiento.

En cuanto a posteriores trabajos, sería muy interesante realizar futuras versiones de la aplicación desarrollada en este proyecto. Se podrían incluir nuevos juegos, así como mejorar la eficiencia e interfaz gráfica de los ya construidos. Dar más libertad al terapeuta a la hora de configurar la aplicación para que pueda trabajar con cualquier movimiento y añadir las imágenes y sonidos que considere adecuadas.

Bibliografía

[1] Xuedong Huang, Alex Acero, Hsiao-Wuen Hon. *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. 2001 by Prentice Hall.

[2] Cambridge University Engineering Department. *The HTK Book*. 2009 Version 3.4.

[3] Tutorial C++ `http://www.cplusplus.com/doc/tutorial/`

[4] Jan Bodnar. *The Winapi (C Win32 API, No MFC) tutorial*. 2007 - 2008

[5] Visual C++ `http://msdn.microsoft.com/library/ms235630.aspx`

[6] Memoria proyecto fin de carrera Carlos Vaquero Avils-Casco, *Reconocedor de Comandos Orales para Eliminar Barreras de Comunicacin y Movilidad en Personas con Discapacidades Motrices y de Comunicación*, Universidad de Zaragoza, 2006.

[7] Indian TEX Users Group. *LATEX Tutorials* 2003 September by A PRIMER.

[8] S. Akyol and P. Alvarado, *Finding Relevant Image Content for mobile Sign Language Recognition*, Department of Technical Computer Science RWTH Aachen, Germany, 2001.

[9] M.-H. Yang, N. Ahuja, and M. Tabb, *Extraction of 2D Motion Trajectories and Its Application to Hand Gesture Recognition*, IEEE Trans. Pattern Analysis Machine Intelligence, vol. 24, no. 8, pp. 1061-1074, Aug. 2002.

[10] E.-J. Ong and R. Bowden, *A Boosted Classifier Tree for Hand Shape Detection*, Centre for Vision, Speech and Signal Processing University of Surrey, Guildford, 2004.

[11] N. Tanibata, N. Shimada, and Y. Shirai, *Extraction of Hand Features for Recognition of Sign Language Words*, Computer-Controlled Mechanical Systems, Graduate School of Engineering, Osaka University 2002.

[12] F.-S. Chen, C.-M. Fu, and C.-L. Huang, *Hand Gesture Recognition Using a Real-Time Tracking Method and Hidden Markov Models*, Image and Vision Computing 21, pp. 745-758, 2003.

[13] Alana Da Gama, Thiago Chaves, Lucas Figueiredo, Veronica Teichrieb, *Poster: Improving Motor Rehabilitation Process through a Natural Interaction Based System Using Kinect Sensor*, Voxar Labs at Center of Informatics of the Federal University of Pernambuco, 2012.

[14] K. K. Biswas, Saurav Kumar Basu, *Gesture Recognition using Microsoft Kinect*, 2011.

[15] Orasa Patsadu, Chakarida Nukoolkit and Bunthit Watanapa, *Human Gesture Recognition Using Kinect Camera*, School of Information Technology King Mongkuts University of Technology Thonburi, Bangkok, Thailand, 2012.

[16] Yao-Jen Changa, Shu-Fang Chenb, Jun-Da Huangc, *A Kinect-based system for physical rehabilitation: A pilot study for young adults with motor disabilities*, Volume 32, Issue 6, Pages 2566-2570, 2011.

Índice de figuras

Apéndice A

Implementación de la aplicación en C++

A.1 Funciones para trabajar con matrices en C++

Para construir el reconocedor en lenguaje C++, que es el lenguaje de programación usado en el SDK de Kinect, necesitamos implementar los tipos matriz, Gaussiana y mezcla de Gaussianas,

```
typedef struct{
  float **matriz;
  int r,c;
} Matrix;

typedef struct{
    Matrix* mean;
    Matrix* cov;
    double aux;
} Gauss;

typedef struct{
    int dimension;
    int n_components;
    Matrix* pc;
    Gauss* gauss[N_COMPONENTS];
    Matrix* X;
} PDF_MFGauss;
```

Creamos también las siguientes funciones básicas para el procesado de matrices en C++:

- `Matrix* initiate_matrix(int r, int c)` ⇒ Inicializa matriz con el número de filas y columnas especificado
- `void free_matrix(Matrix *M)` ⇒ Libera el espacio de memoria ocupado por la Matriz M

- void set_value_matrix(Matrix *M, float value) ⇒ Pone value en todos los valores de la matriz M
- void matrix_get_column(Matrix *M,int c,Matrix *M_col) ⇒ Copia la columna c de la matriz M a la matriz M_col
- void copy_col_matrix(Matrix *M, Matrix *M_col, int c) ⇒ Copia la matriz M_col en la columna c de la matriz M
- Matrix* sub_matrix(Matrix *M1,Matrix *M2) ⇒ Resta 2 matrices
- void mult_matrix_scalar(Matrix *M, float x) ⇒ Multiplica la matriz M por el número x
- Matrix* mult_matrix(Matrix *M1,Matrix *M2) ⇒ Multiplica 2 matrices
- void div_matrix_dot(Matrix *M1,Matrix *M2, Matrix *M) ⇒ Divide 2 matrices punto a punto
- Matrix* t_matrix(Matrix *M) ⇒ Calcula la traspuesta de la matriz M
- Matrix* inv_matrix(Matrix *M) ⇒ Calcula la inversa de la matriz M
- double det_matrix(Matrix *M)) ⇒ Calcula el determinante de una matriz

A.2 Windows API

Para implementar la interfaz de usuario, utilizamos las funciones API de Windows. [4]

En primer lugar, construimos la pantalla principal de nuestra aplicación. En la función *WndProc* procesamos los mensajes generados:

- WM_CREATE: Al inicializar la ventana principal, se crean los botones de acceso a los distintos apartados de la aplicación (el entrenamiento y los 4 juegos). Al pulsar cada uno de estos botones, se crea una nueva ventana que depende de la principal. Para ello, se utiliza la técnica de **subclasificación** que se explica más adelante en este mismo anexo.
- WM_COMMAND: Creamos el Menú de la ventana principal. Desde aquí, el usuario podrá seleccionar el directorio de trabajo, configurar la visualización del vídeo y de las posiciones del esqueleto en los juegos y salir de la aplicación.
- WM_PAINT: Este mensaje se recibe cada vez que se maximiza o minimiza la ventana, cuando se la redimensiona, o se la vuelve a mostrar

luego de estar tapada por otra ventana. Por tanto, desde aquí mostramos los dibujos de los juegos en la ventana principal.

Método a utilizar

1. Cargamos las imágenes en el proyecto en formato BMP y declaramos el nombre con el que vamos a identificarlas,

   IMG BITMAP "img.bmp"

2. Creamos un "handle"a la imagen IMG.
3. Creamos un "Device Context"llamado hdcMem el cual se enlaza al handle de la imagen. Los "Device Context"son áreas de memoria donde podemos dibujar y copiar imágenes.
4. Se llama a GetObject para obtener las dimensiones de la imagen que se guardan en el objeto BITMAP bm.
5. La imagen está cargada en la memoria RAM. Para que aparezca en la ventana debemos copiarla. Eso se hace con la api BitBlt cuyos parámetros son: destino (el handle del Device Context de la ventana, que es el valor devuelto por la api BeginPaint y guardado en la variable hdc), coordenada "x"donde se ubicará la esquina superior izquierda de la imagen copiada, coordenada "y"donde se ubicará la esquina superior izquierda de la imagen copiada, ancho de la imagen, alto de la imagen, handle del Device Conext con la imagen a copiar, coordenada "x"de la esquina superior izquierda de la imagen a copiar, coordenada "y"de la esquina superior izquierda de la imagen a copiar, operación de copiado.
6. Por último, debemos borrar de la memoria los handles, bitmaps y demás objetos.

```
PAINTSTRUCT ps;
hdc = BeginPaint(hWnd, &ps);
BITMAP bm;
//Se crea handle a la imagen
HBITMAP himg = LoadBitmap (hInstance, MAKEINTRESOURCE (IMG));
//Se crea Device Context
HDC hdcMem = CreateCompatibleDC(hdc);
//Se enlaza el Device Context al handle de la imagen
HBITMAP hbmOld = (HBITMAP) SelectObject (hdcMem, himg);
//Se obtienen las dimensiones de la imagen
GetObject(himg,sizeof(bm),&bm);
//Se copia la imagen de la memoria RAM en la ventana
BitBlt(hdc,500,50,bm.bmWidth,bm.bmHeight,hdcMem,0,0,SRCCOPY);
//Borrar handles, bitmaps y dem s objetos
DeleteObject(himg);
SelectObject(hdcMem,hbmOld);
DeleteDC(hdcMem);
```

```
DeleteObject(hbmOld);
EndPaint(hWnd, &ps);
```

Subclasificación

La técnica de subclasificación permite cambiar la dirección del procedimiento de ventana. De esta forma, podemos procesar los mensajes que recibe la nueva ventana o control y pasar los restantes al procedimiento de la ventana principal.

Método a utilizar

1. Se almacena el puntero this en los datos de usuario. Para ello, se llama a SetWindowLongPtr con el parámetro GWLP_USERDATA.
2. Cambiar el procedimiento de ventana por defecto por la función DispatchProc (sin puntero this). Esto se hace utilizando SetWindowLongPtr con el parámetro GWLP_WNDPROC que devolverá el puntero al antiguo procedimiento: ViejoProc.
3. Crear la función Dispatch que se encargará de obtener el puntero this almacenado en el paso 1 con GetWindowLongPtr y llamará al procedimiento de mensajes adecuado.
4. Procesar los mensajes.
5. Procesar los mensajes en su procedimiento por defecto. Se utiliza CallWindowProc con el antiguo procedimiento devuelto en el paso 2.

```
void boton_gestos(HWND hWindow, UINT x, UINT y, UINT w, UINT h)
{
    hCONTROL = 0;
    HWND hPADRE = hWindow;

    if (!hCONTROL)
        hCONTROL = CreateWindowEx (NULL,
                        TEXT("BUTTON"),
                        TEXT("Gestos"),
                        WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
                        x, y, w, h,
                        hPADRE,
                        NULL, NULL, NULL);
        if (hCONTROL == NULL)
            MessageBox(hPADRE,
                    TEXT("Ocurrió un error al crear el control."),
                    TEXT("Error"), MB_OK | MB_ICONERROR);
        else
        {
         //Cambiar fuente del control
```

```
            SendMessage(hCONTROL, WM_SETFONT,
                        (WPARAM) GetStockObject(DEFAULT_GUI_FONT),
                        MAKELPARAM(FALSE, 0));
//1.Guardar el puntero this en los datos de usuario del control
            SetWindowLongPtr(hCONTROL, GWLP_USERDATA,
                             (LONG_PTR) this);

//2. Subclasificar, cambiar el procedimiento de ventana por
//   defecto en el control por DispatchProc (sin puntero this)
            ViejoProc = (WNDPROC) SetWindowLongPtr(hCONTROL,
                                                   GWLP_WNDPROC,
                                                   (LONG_PTR) DispatchProc);
        }
}

//3. Obtener el puntero this almacenado en el paso 1 y
//   llamar al Procedimiento de Mensajes adecuado
static LRESULT CALLBACK DispatchProc(HWND hwnd,UINT iMensaje,
                                     WPARAM wParam,LPARAM lParam)
{
    CSkeletalViewerApp* pControl;
    pControl = (CSkeletalViewerApp *) GetWindowLongPtr(hwnd,
                                                       GWLP_USERDATA);

    //Despachar a procedimiento correspondiente al control
    return pControl->Procedimiento (hwnd, iMensaje,
                                    wParam, lParam);
}

//4. Procedimiento de Ventana
LRESULT CALLBACK Procedimiento(HWND hwnd, UINT iMensaje,
                               WPARAM wParam, LPARAM lParam)
{
    if (iMensaje == WM_LBUTTONUP)
    {
        //Creamos la ventana del juego gestos
    }
//5. Procesar mensajes en procedimiento de ventana por defecto
    return CallWindowProc (ViejoProc, hwnd, iMensaje,
                           wParam, lParam);
}
```

A.3 SkeletalViewer

El sensor Kinect incluye cámaras que capturan información de la profundidad y los colores de la imagen, así como los movimientos del esqueleto. La interfaz de usuario (NUI API) del SDK de Kinect, permite el acceso y la manipulación de estos datos. Para construir el reconocedor, utilizaremos la clase CSkeletalViewerApp:

INICIALIZACIÓN DEL SENSOR KINECT

Los métodos Nui_Zero y Nui_Init inicializan el sensor Kinect y empieza el procesado de los datos en tiempo real:

- **Crea un evento que está asociado a cada tipo de datos:**

```
m_hNextDepthFrameEvent = CreateEvent( NULL, TRUE, FALSE, NULL );
m_hNextColorFrameEvent = CreateEvent( NULL, TRUE, FALSE, NULL );
m_hNextSkeletonEvent = CreateEvent( NULL, TRUE, FALSE, NULL );
```

- **Inicializa las estructuras para la representación del esqueleto.**
  SkeletalViewer dibuja el esqueleto usando el subsistema Windows GDI (Graphics Device Interface).
  SkeletalViewer dibuja una imagen del esqueleto en una región de memoria y luego pasa un puntero a GDI para mostrar la imagen generada en el dispositivo.

- **Inicializa las estructuras para la representación de imágenes.**
  SkeletalViewer usa Direct3D 9 para representar las imágenes de profundidad y de video. Usando Direct3D 9, una aplicación puede configurar varios buffers, cada uno de los cuales contiene un fotograma individual. La aplicación llama a Direct3D 9 para mostrar los fotogramas secuencialmente intercambiando los punteros de los buffers, por tanto, generando una transferencia de bloques de bits (blt).

  La clase DrawDevice (definida en el fichero *DrawDevice.cpp* del SDK de Kinect) tiene métodos que crean y manipulan las estructuras Direct3D 9. En la clase CSkeletalViewerApp se declaran los objetos DrawDevice para las imágenes de profundidad y video, m_DrawDepth y m_DrawVideo respectivamente. CSkeletalViewerApp::Nui_Init crea e inicializa los dispositivos y mapas de bits de Direct3D 9 que la aplicación usa con cada uno de estos objetos.

- **Inicializa el sensor Kinect.**
  Una aplicación debe inicializar el sensor Kinect llamando a NuiInitialize que recupera los datos del sensor Kinect y las señales de la aplicación cuando se detectan las imágenes. Luego, Nui_Init permite el seguimiento del esqueleto llamando a NuiSkeletonTrackingEnable. Cuando está activado el seguimiento del esqueleto, se procesan los datos de imagen y profundidad para entregar imágenes que incluyen datos del esqueleto. Se puede habilitar o deshabilitar el seguimiento del esqueleto en cualquier momento durante el procesado.

- **Abre streams para las imágenes de color y profundidad.**
  Una vez que se completa la inicialización del sensor, Nui_Init abre

streams de salida para los datos del color y profundidad llamando a NuiImageStreamOpen.

- **Crea el hilo de procesamiento de datos del sensor Kinect.**
  La función CreateThread crea un nuevo hilo e immediatamente empieza a ejecutar el método Nui_ProcessThread de la clase CSkeletalViewerApp.

PROCESADO DE LOS DATOS CAPTURADOS POR EL SENSOR KINECT

El sensor Kinect suministra las tramas de datos en intervalos regulares tanto si una aplicación está esperando a recibirlos o no.
El procesado de los datos del sensor en *SkeletalViewer* está controlado por un bucle en el método CSkeletalViewerApp::Nui_ProcessThread

```
while ( continueProcessing )
// Wait for event.
// If the stop event occurs, stop looping and exit
// Blank the skeleton display.
// Process frame events
switch(EventType) {
        case Depth:
            Nui_GotDepthAlert();
            break;
        case Video:
            Nui_GotVideoAlert();
            break;
        case Skeleton:
            Nui_GotSkeletonAlert( );
            break;
}
```

Cuando un evento de profundidad de imagen ocurre, el método Nui_ProcessThread llama a Nui_GotDepthAlert para procesar los datos de profundidad. En el caso del video, se llama Nui_GotVideoAlert. Estos métodos devuelven la imágenes de profundidad y video y las representan en la pantalla usando la clase DrawDevice. Cuando un evento de esqueleto ocurre, el método Nui_ProcessThread llama a Nui_GotSkeletonAlert que procede de la siguiente forma:

1. **Recupera una imagen de los datos del esqueleto llamando a NuiSkeletonGetNextFrame.**

2. **Comprueba si la imagen describe un esqueleto y si no termina.**
   Cada vez que se procesa una imagen de profundidad ocurre un evento de esqueleto, así que es posible recuperar la imagen del esqueleto sin contener datos de esqueleto. Si la imagen no contiene un esqueleto, Nui_GotSkeletonAlert termina.

3. **Arregla la imagen para mostrarla llamando a NuiTransformSmooth.**
   La representación secuencial de las imágenes del esqueleto puede dar como resultado imágenes que parpadeen. Para reducir esto, Nui_GotSkeletonAlert llama a NuiTransformSmooth para aplicar un filtro de alisado a los datos del esqueleto.

4. **Guarda las coordenadas de las posiciones del esqueleto.**
   En la estructura SkeletonFrame.SkeletonData.SkeletonPositions[j] se almacena la posicin (x, y, z) de 20 puntos del esqueleto (figura 2.2). Guardamos esta información en una matriz de dimensiones 50x60 (50 filas, una por cada captura de posiciones y 60 columnas - 20 puntos del esqueleto x 3 coordenadas).

```
float matriz_esqueleto [50] [60] ;
for (int j = 0; j < 20; j++)
        {   matriz_esqueleto [i] [l]    =
            SkeletonFrame.SkeletonData.SkeletonPositions[j].x;
            matriz_esqueleto [i] [l+1] =
            SkeletonFrame.SkeletonData.SkeletonPositions[j].y;
            matriz_esqueleto [i] [l+2] =
            SkeletonFrame.SkeletonData.SkeletonPositions[j].z;
            l = l + 3;
        }
```

5. **Dibuja el esqueleto llamando a Nui_DrawSkeleton.**

6. **Muestra la imagen representada en la pantalla llamando a Nui_DoDoubleBuffer.**

SALIR DE LA APLICACIÓN

Cuando el usuario cierra la ventana, se envía el mensaje WM_CLOSE desde donde llamamos a la función DestroyWindow(hWnd). Ésta genera el mensaje WM_DESTROY desde donde llamamos a Nui_UnInit para borrar los objetos creados y liberar memoria.

Apéndice B

Diagrama de Gantt

A continuación, se presentan las tareas llevadas a cabo en este proyecto. En la figura B.1 se muestra el Diagrama de Gantt teniendo en cuenta la duración de cada tarea en caso de que la dedicación al proyecto hubiese sido a tiempo completo.

1. Documentación
    - Textos científicos basados en el reconocimiento
    - Algoritmo de Viterbi, Modelos ocultos de Markov (HMMs)
    - Kit de desarrollo de Software Kinect[1]
2. Base de datos
    - Función para guardar las coordenadas del esqueleto en un fichero de texto
    - Grabación de movimientos con el brazo derecho
3. Prototipo Matlab
    - Función para cargar los ficheros de entrada y calcular los ángulos del cuerpo
    - Función para entrenar los HMMs
    - Función para realizar el reconocimiento
4. Prototipo C++[2]
    - Creación de las estructura de datos
    - Implementación de las funciones creadas en Matlab

[1] `https://dev.windows.com/en-us/kinect`
[2] `http://www.cplusplus.com/doc/tutorial/`

5. Análisis prestaciones
    - Pruebas de reconocimiento en Matlab
    - Reconocedor en C++: Tasas de acierto y error
6. Aplicación: Juegos
    - Interfaz de usuario (funciones API de Windows[4])
    - Menús de configuración
    - Entrenamiento
    - Juegos: Gestos, Adivinanzas, Frases y Evocación
7. Memoria
    - Latex[3]

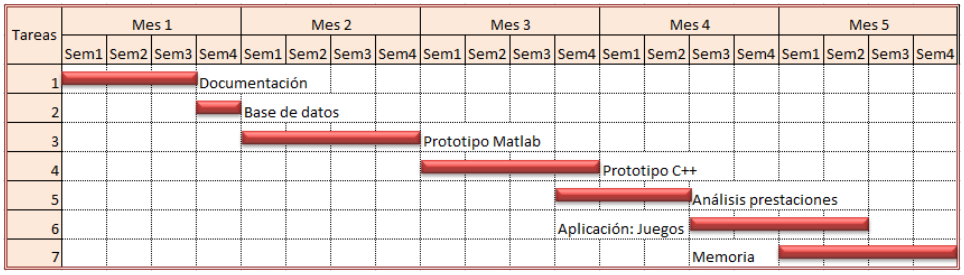


Figura B.1: Diagrama de Gantt

[3] `http://www.latex-project.org/`