UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Generación de lengua de signos continua

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Aitana Villaplana Moreno

Tutor: Carlos David Martínez Hinarejos

Curso 2019 - 2020

Agradecimientos

Agradezco a todas las personas que me ayudaron a llegar hasta aquí. A mi tutor Carlos David Martínez Hinarejos, que me guió durante todo el proyecto. A mi familia por permitirme cursar estos estudios y apoyarme. A mi mejor amigo Javier Garrido por animarme y ayudarme cuanto sabía. Y todas las personas que directa o indirectamente me ayudaron a llegar hasta aquí. Gracias.

Resum

En aquest treball s'aborda el problema de la generació de frases en llenguatge de signes, a partir de paraules soltes (específicament usant la llengua de signes espanyola). A partir de les dades de les paraules, extretes mitjançant el sensor *Leap Motion*, es genera una frase contínua equivalent a la que es podria extreure del mateix sensor.

Paraules clau: Intel·ligència artificial, llengua de signes, Models Ocults de Markov, *Leap Motion*, Interpolació lineal

Resumen

En este trabajo se aborda el problema de la generación de frases en lenguaje de signos, a partir de palabras sueltas (específicamente usando la lengua de signos española). A partir de los datos de las palabras, extraídas mediante el sensor *Leap Motion*, se genera una frase continua equivalente a la que se podría extraer del mismo sensor.

Palabras clave: Inteligencia artificial, lengua de signos, Modelos Ocultos de Markov, *Leap Motion*, Interpolación lineal

Abstract

This work deals with the problem of the generation of sentences in sign language, from single words (specifically using Spanish sign language). From the data of the words, extracted using the *Leap Motion* sensor, a continuous sentence equivalent to that which could be extracted from the same sensor is generated.

Key words: Artificial intelligence, sign language, Hidden Markov Models, *Leap Motion*, Linear interpolation

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

CAPÍTULO 1
Introducción

La lengua de signos es un lenguaje que usa gestos con manos y brazos, que generalmente sirve para comunicarse con personas con discapacidad auditiva o dificultad en el habla.

Cada gesto representa una palabra del lenguaje, y se van formando oraciones enlazando estos gestos. No existe una lengua de signos universal, hay una variedad de lenguas de signos dependiendo de la zona y del lenguaje oral que se utilice. En España, están reconocidas oficialmente la lengua española y la catalana.

Esta lengua es especialmente importante para integrar a todas las personas con problemas auditivos o del habla en la comunicación; por eso se está trabajando en el ámbito de la Informática para facilitar lo máximo posible la integración de estas lenguas en la comunicación, la tecnología y la sociedad. Hoy en día ya existen técnicas de inteligencia artificial que se utilizan para reconocimiento de lengua de signos y traducciones [1] [2] [3].

En este trabajo se plantea resolver un problema de generación de frases en lenguaje de signos a partir de palabras aisladas. Disponemos de una serie de palabras en lenguaje de signos. Estas palabras fueron representadas en lengua de signos, grabadas y, posteriormente, pasadas a datos numéricos. También disponemos de un cierto número de frases formadas con dichas palabras.

El trabajo a realizar sería conseguir, mediante un programa, formar estas frases a raíz de las palabras aisladas que la forman, como si las representara una persona en lengua de signos. Es decir, se trata de enlazar las palabras de la frase como se haría de forma natural.

Dado que las palabras aisladas se empiezan a representar desde un estado de reposo (la o las manos quietas y por debajo de la cintura) y finalizan de la misma forma, la idea básica del trabajo sería eliminar esta fase de reposo entre palabra y palabra, es decir, conseguir que la frase se diga de forma fluida, teniendo en cuenta la posición de la palabra anterior para comenzar a representar la siguiente, tal y como se haría de manera natural.

1.1 Motivación

El motivo fundamental de la elección del tema del trabajo es la motivación por realizar un trabajo enfocado en la rama de Computación, y así poder aplicar los conocimientos adquiridos a través del grado, en especial los de la rama de Computación.

También reforzar y practicar varios conceptos y técnicas utilizadas anteriormente en trabajos relacionados con el tema del TFG, como, por ejemplo, los Modelos Ocultos de Markov (HMM) [1] [2] o las Redes Neuronales [3]. Este fue el motivo principal de la selección del tema del trabajo.

Como motivación adicional, podemos añadir también el interés altruista de realizar un avance en la investigación de herramientas para informatizar y digitalizar el lenguaje de signos, y así ayudar en menor o mayor medida a las personas que utilicen este lenguaje en su comunicación.

1.2 Objetivos

Los objetivos que se esperan conseguir son los siguientes:

- Realizar un estudio de las posibles formas de resolver el problema a tratar.
- Conocer y comprender las técnicas utilizadas anteriormente en otros trabajos relacionados.
- Ampliar los conocimientos de las mismas.
- Seleccionar la mejor opción para realizar nuestro trabajo en base a ciertos parámetros (error y coste, entre otros).
- Implementar el código de manera eficiente y minimizando el coste computacional.
- Crear una buena documentación para que cualquiera ajeno al trabajo pueda consultarlo.

Estos serían los objetivos básicos que trataremos de conseguir durante la realización del trabajo.

1.3 Estructura de la memoria

Procedemos a explicar más en detalle la estructura que seguiremos en la memoria, y los apartados en los que dividimos el trabajo realizado.

Primero, pasamos a hablar del estado del arte, donde presentamos la situación actual de ámbito en el que vamos a trabajar, los trabajos realizados relacionados con el tema y los resultados de los mismos. Una vez dicho eso, se procederá a explicar nuestra propuesta de trabajo, teniendo en cuenta estos trabajos anteriores.

En segundo lugar, pasaremos a explicar y analizar el problema presentado, y a identificar posibles soluciones del mismo. Una vez hecho esto, se presentará la solución propuesta en base a los criterios que se consideren oportunos, tales como el coste computacional, la complejidad de la solución o la factibilidad de la misma, entre otros.

A continuación, presentaremos el diseño de la solución, mediante su arquitectura, su estructura y el diseño detallado, así como la tecnología utilizada para llegar a dicha solución, si es que se requiere. En este apartado se explicará con detalle la estructura de los datos con los que se trabaja, y el código en detalle con cada una de las funciones utilizadas, así como clases o atributos.

También presentaremos el desarrollo de la solución, tanto la entrada de datos del sistema como cada una de las fases del desarrollo de la solución, con los algoritmos que se crean convenientes para explicar el desarrollo.

Tras ello se pasará a presentar las diferentes pruebas realizadas para comprobar el correcto funcionamiento de la solución presentada, así como otros parámetros interesantes como el coste computacional. Para finalizar, se pasará a presentar las conclusiones del trabajo y también la relación de éste con los estudios cursados del grado.

CAPÍTULO 2
Estado del arte

En este apartado procedemos a explicar la situación actual de la tecnología de reconocimiento y traducción del lenguaje de signos a lenguaje natural escrito.

Actualmente existe una tecnología capaz de realizar el reconocimiento de estos datos, como, por ejemplo, el sensor *Leap Motion*[1], comentado anteriormente. El aparato (que dispone de un sensor), se coloca en una posición fija y a una distancia determinada de las manos de la persona que realiza la representación de la o las palabras correspondientes a reconocer. Otro sensor muy utilizado en la actualidad es Microsoft Kinect.[2]

En la sección 4.1, se explicará más en detalle el funcionamiento de *Leap Motion*, utilizado en varios casos, y el formato de salida de los datos.

Partiendo de la información obtenida mediante el sensor, se han realizado una serie de trabajos para su reconocimiento. En primer lugar, tal y como se cita en el artículo [1], se han utilizado varias técnicas como redes neuronales (NN) A.1 y mixturas gaussianas (GMM) A.2, obteniendo muy buenos resultados, con un bajo porcentaje de error en el reconocimiento de caracteres (1,85 % y 5,8 %, respectivamente).

En [1] y [2], podemos observar que el trabajo de reconocimiento se realizó mediante Modelos de Markov Ocultos (HMM) A.3, aplicado tanto a reconocimiento de una palabra como al de oraciones. Los experimentos realizados en [1] se basaron principalmente en clasificar las palabras individuales y el reconocimiento de las oraciones. Para la clasificación de palabras se realizaron experimentos con HMM con topología fija, es decir, un número de estados fijo, y con topología variable, donde varía el número de estados según la longitud de la palabra y las transiciones. Como era de esperar, los resultados realizados con topología variable eran ligeramente mejores, a pesar de ser bastante similares (alrededor de 10 % de error en el mejor de los casos). Para el reconocimiento de oraciones se obtuvieron resultados muy similares, también con una ligera ventaja de la topología variable frente a la fija (con un 12 % de error aproximadamente). Por tanto, se obtuvieron unos resultados bastante prometedores en cuanto a ratio de error se

[1] https://www.ultraleap.com/product/leap-motion-controller
[2] https://developer.microsoft.com/es-es/windows/kinect/

refiere.

En [2] se comparan dos técnicas, la primera utilizando HMM continuos, es decir, con datos de magnitud continua. En esta ocasión se utilizó el Hidden Markov Model Toolkit (HTK)[3][10], una herramienta utilizada para manipular el HMM de forma que se ajusten mejor los parámetros al problema en cuestión. En el mejor de los casos, con 8 estados, 7 iteraciones y dos mixturas Gaussianas, se obtuvo una precisión del 87,4 % en el caso del reconocimiento de palabras, frente a un 61,3 % para las oraciones.

Para la segunda técnica, el proceso de reconocimiento se divide en dos partes. La primera, donde se reconocía la forma inicial de la mano antes de comenzar a hacer el movimiento, se realizó con *K-Nearest Neighbour* (KNN) A.4. La segunda parte, donde se seguía el movimiento de la mano y se reconocía el gesto, se obtuvo mediante *Dynamic Time Warping* (DTW) A.5 [7]. Con esta técnica se obtuvo una precisión del 88,4 % en el mejor de los casos.

Comparando ambas técnicas, se obtiene un ratio de precisión bastante similar; sin embargo, el tiempo de cómputo para el HMM es de 519 segundos, frente al KNN + DWT con 9383 segundos, lo que lleva a los autores a escoger HMM como mejor opción.

Otra de las técnicas muy utilizadas son las Redes Neuronales (NN). En el caso de [3], se utilizó la Red Neuronal Convolucional (CNN) A.6 de LeNet [8], y se realizaron experimentos variando el máximo número de filas (los *frames* en los que se captura el gesto) y el número de iteraciones. Se realizó también un preproceso antes de pasar al reconocimiento. Los mejores resultados se obtuvieron utilizando la técnica de interpolación como preproceso, un número máximo de filas de 97 y un número máximo de iteraciones de 25, con un error del 8,6 %.

Otro ejemplo del uso de las NN, donde se utilizaron como datos la lengua italiana de signos, es presentado en [4]. Los experimentos se realizaron mediante la extracción de datos con Microsoft Kinect, y se utilizan también las CNN, además de otras Redes Neuronales Artificiales, junto con un preproceso previo. Se utilizaron CNN de tres capas de profundidad.

El resultado del preproceso fueron 4 vídeos, dos de ellos de la mano a analizar en cuestión, donde uno de ellos es un mapa de profundidad (en el cual se ha reducido el ruido y se ha eliminado el fondo), y otro con escala de grises. Los otros dos son del cuerpo completo, con las mismas características que los anteriores.

La arquitectura está formada por dos redes CNN, que se encargan de extraer las características, y otra NN que se encarga de la clasificación. Cada CNN recibe los dos vídeos de la mano y el cuerpo completo, respectivamente, y el resultado de estos pasa a la NN, que proporciona el resultado. En el mejor de los casos se obtuvo un error del 8,3 %.

[3] http://htk.eng.cam.ac.uk/

También se han realizado trabajos de traducción del alfabeto; en el caso de [5] se hizo sobre Lengua de Signos Colombiana, donde se utilizó un dispositivo FPGA; este dispositivo venía formado por las siguientes etapas: captura de imagen, preprocesamiento, extracción de características y reconocimiento.

La captura de imagen se realizó con una cámara digital. El preproceso se realizó mediante hardware, cuyo resultado es una imagen con la mano en blanco y el fondo en negro (imagen binaria). La extracción de características se realiza mediante dos vectores de transformación que representan la mano; éstos se concatenan y forman el vector de características. Finalmente, el reconocimiento se basa en una NN perceptrón Multicapa (MLP) A.7 [12], formada por una capa de entrada, una capa oculta con 90 neuronas, y una de salida. Se obtuvieron unos resultados con el 98,2 % de acierto.

2.1 Crítica al estado del arte

En la actualidad, se ha conseguido realizar el reconocimiento de palabras y letras en lengua de signos con un error bastante tolerable. Sin embargo, hay una serie de puntos flacos, ya que a la hora de reconocer frases el error de reconocimiento pasa a ser algo significativo. Esto impide que estas técnicas se utilicen en conversaciones reales o casos donde la comunicación sea crítica.

La causa de este problema reside principalmente en que los sistemas de reconocimiento de lengua continua (frases) necesitan una gran cantidad de datos para poder entrenar los modelos y perfeccionarlos, de manera que se puedan hacer más precisos y minimizar el error de reconocimiento. Esto se dificulta mucho debido al gran coste que supone obtener tales datos, ya que existen muchas combinaciones de palabras a la hora de formar frases, por lo que los datos obtenidos mediante el sensor son excesivamente limitados.

Otra de las dificultades viene dada por la gran variedad de lenguas de signos que hay en el mundo, lo que dificulta mucho crear un sistema reconocedor independiente del idioma. Esto nos obliga a realizar un caso específico para cada lengua.

2.2 Propuesta

La propuesta de este TFG se basa en conseguir formar frases fluidas en lengua de signos a partir de las palabras aisladas que forman la frase, lo que nos llevaría a poder formar frases a partir de palabras.

El objetivo fundamental del trabajo es generar un conjunto de datos de gran tamaño con frases continuas, algo que no está disponible debido al alto coste que supone grabar múltiples combinaciones posibles de palabras para formar frases, además del coste que supondría grabar a numerosas personas interpretando tal cantidad de frases. Por lo tanto, este trabajo nos permitirá aumentar enormemen-

te el número de datos, lo que nos llevaría a poder estimar modelos estadísticos más complejos. Es decir, la creación de estas frases sintéticas nos permite obtener modelos más efectivos a la hora de hacer el reconocimiento de la lengua de signos continua, ya que estos modelos necesitan una cantidad masiva de datos para estimarse bien y esta cantidad no está disponible por el momento.

Como objetivos adicionales, podemos añadir que el trabajo también serviría para facilitar la comunicación entre una persona con discapacidad auditiva y otra sin ella. Se podría utilizar como parte de un sistema en el cual la persona oyente introduciría las palabras con las que desea formar la frase y el sistema realizaría la representación formando una frase continua en lengua de signos, obtenida mediante este trabajo. Esto daría mayor sensación de realismo y comunicación a la otra persona.

Esto también aportaría una gran ventaja con respecto a los trabajos realizados anteriormente, dado que sabemos que el reconocimiento de palabras sueltas es muy eficaz, a diferencia del de las frases, que todavía presenta un error considerable. En este caso nos aprovechamos de ese bajo error a nivel de palabra para realizar la conversión de palabra a frase, sin generar un alto porcentaje de error.

CAPÍTULO 3
Análisis del problema

Tal y como hemos introducido en el apartado anterior, el problema a resolver en este trabajo es crear una aplicación que nos realice el paso de palabras sueltas a frases continuas, es decir, enlazando las palabras en vez de tener palabras representadas aisladamente.

El problema fundamental de esto reside en que cada palabra parte de un estado de reposo (sin realizar movimiento) y finaliza también en este estado de reposo. Esto provoca que no sirva simplemente concatenar palabras para formar la frase, ya que así habría simplemente palabras sueltas representadas una detrás de otra, lo que produciría en la representación una falta de expresividad y fluidez, además de no ser realista.

El objetivo de este trabajo es pues, en primer lugar eliminar este estado de reposo entre palabras; es decir, en una subsecuencia de dos palabras dentro de la frase, eliminar el reposo del final de la primera palabra y el del inicio de la segunda. Además, también es necesario realizar el movimiento intermedio del paso de una palabra a otra, según como queda la posición de la mano en la palabra primera, tal y como se haría en la realidad.

Finalmente, se debe calcular el error de reconocimiento de las frases artificiales en las mismas circunstancias en las que se reconocerían las frases originales, y calcular el error de reconocimiento para comprobar que son frases que podrían ser utilizadas de la misma forma que las originales.

En conclusión, este problema se puede dividir en dos subproblemas principales. El primero de todos, y el más complicado, se trata de averiguar de cada palabra qué *frames* (vectores de características que forman la palabra) corresponden al reposo inicial, cuáles al final y cuáles a la representación propiamente dicha de la palabra; la resolución de este problema se tratará más adelante en la sección 5.3. Este punto es el más difícil, debido a que no existe ningún tipo de información sobre el reposo de las muestras, ni existe ninguna forma de comprobar que se ha calculado correctamente esta parte, por lo que no hay forma exacta de comprobar que esa parte de nuestra solución es correcta, de forma que se intentará buscar una resolución que seleccione un par de pequeñas secuencias, una al inicio y otra al final de cada palabra, y obtener una solución aproximada a este subproblema.

El otro subproblema es cómo generar los puntos intermedios entre palabras, una vez eliminados estos estados de reposo; estos puntos se generarán mediante una o varias técnicas de interpolación que se explicarán en detalle en la sección 5.4. Una vez hecho esto, se realizarán las pruebas pertinentes para comprobar la solución.

3.1 Solución propuesta

La solución que se propone es, en primer lugar, seleccionar el modo de calcular el reposo de la manera más exacta posible y, una vez hecho esto, buscar una manera de interpolar los puntos intermedios de una manera realista.

En primer lugar se eligió un lenguaje de programación adecuado para este tipo de problemas y, debido a la naturaleza del trabajo, se seleccionó Python ya que es uno de los más utilizados en el ámbito matemático, de inteligencia artificial y sus numerosas librerías para el manejo de matrices, como es el caso de este trabajo.

CAPÍTULO 4
Diseño de la solución

Procedemos a explicar el diseño de la solución, así como las herramientas necesarias para realizarla, el funcionamiento del sensor, los datos obtenidos y su estructura, así como la transformación y proceso que siguen para convertirse en datos de entrada de la solución. También se explicará con detalle el funcionamiento del código, el lenguaje elegido y la estructura software seguida.

Podemos dividir este trabajo en los siguientes pasos: tratamiento de los datos, procesamiento y realización de la solución, y comprobación del error.

En primer lugar, la lectura y tratamiento de los datos se realiza leyendo los ficheros correspondientes a las palabras con las que se quiera formar la frase y pasando sus datos a la estructura de matriz correspondiente, para poder trabajar con los datos con mayor facilidad.

Una vez tenemos estos datos pasaremos a realizar la solución, es decir, pasar de las palabras a la frase fluida a través de realizar una serie de interpolaciones para eliminar los huecos entre las palabras, y enlazarlas mediante movimientos suaves de transición entre ellas. Este proceso nos daría como resultado un archivo similar al de las frases iniciales, ideal para comparar nuestro archivo creado artificialmente con archivos derivados de datos reales.

4.1 Tecnología utilizada

En este apartado procedemos a explicar en detalle el funcionamiento y utilización del sensor *Leap Motion* para la adquisición de los datos utilizados en este trabajo y en otros trabajos nombrados anteriormente.

Dicho sensor, que puede observarse en la Figura 4.1, dispone de un eje de coordenadas fijo y va realizando capturas cada determinados milisegundos, transcribiendo la posición en los 3 ejes de coordenadas de cada uno de los siguientes puntos: un punto central de la palma de la mano, un punto por cada uno de los dedos y la rotación de la palma de la mano en el plano correspondiente. Esto es realizado para ambas manos. En caso de que la palabra se represente con una sola mano (en este caso la mano derecha), la mano ausente sería representada con

Figura 4.1: Imagen del sensor Leap Motion.

ceros en todas las columnas correspondientes a esa mano.

La existencia de palabras de una sola mano trae un problema a la hora de trabajar con los datos, ya que los datos a *null* (0) de la mano no utilizada (siempre la izquierda), acarrean problemas a la hora de entrenar el modelo, debido a su nula variabilidad. Para solucionar este problema, se rellenaron las columnas que estaban inicialmente a 0 con números aleatorios, que podrían ser positivos y negativos, generados a partir de la distribución normal con media de 0 y varianza de 0.01.

En resumen, los datos tendrían una estructura en forma de matriz, donde las filas serían cada una de las capturas realizadas (*frames*) y las columnas corresponderían a cada coordenada del punto capturado.

Esto nos daría como resultado una matriz con un número variable de filas (el número de capturas depende de la duración de la representación de la palabra en cuestión) y un número constante de columnas (42).

La matriz se estructura de manera que las 21 columnas de la izquierda de la matriz corresponden a los datos de la mano izquierda, y las 21 columnas a la derecha a la mano derecha, sumando así el total de 42 columnas.

Las 42 columnas corresponden a las coordenadas x, y, z (cada coordenada en una columna) de la posición de la mano, las coordenadas de la rotación de la mano y las de cada uno de los cinco dedos de la mano. Esto suma un total de 21 columnas por mano, es decir, 42 columnas en total. Estas coordenadas, se capturan mediante el sensor por el uso de dos cámaras infrarrojas con una separación conocida, y tres fuentes de iluminación infrarroja que permiten la detección tridimensional de objetos que pasen por delante del sensor. La Figura 4.2 muestra el esquema de

componentes del sensor y sus escalas.

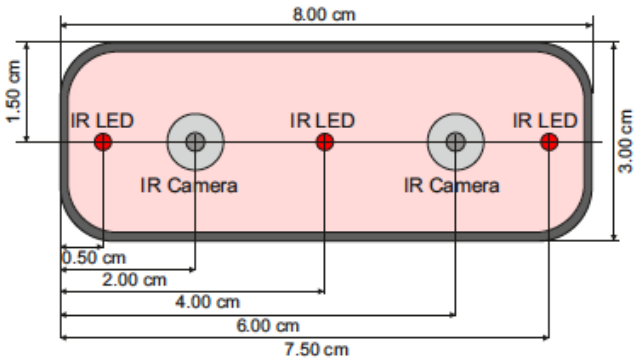

Figura 4.2: Componentes y dimensiones del sensor.

Otra de las tecnologías utilizadas a nivel de programación es una librería que proporciona el lenguaje Python, NumPy [1], que permite tanto creación, como manipulación de vectores y matrices de manera sencilla, así como también de funciones necesarias para desarrollar el proyecto de este TFG. Es el caso de operaciones tales como la concatenación de matrices y la escritura de las matrices en ficheros de texto, entre otras. También fue importante el uso de la librería argparse [2] para realizar la recogida y tratado de los parámetros de entrada del programa.

También se utilizó la librería dtaidistance [3] para realizar las pruebas con DTW [7], tanto para calcular la distancia entre dos frases como para la visualización de las pruebas, explicado con más detalle en el capítulo 6.

4.2 Datos disponibles

Para realizar este trabajo partimos de un conjunto de palabras y frases extraídas mediante el sensor *Leap Motion*.

El conjunto de palabras dispone de un total de 92 palabras representadas por 4 personas cada una, con una repetición de 10 veces cada palabra, lo que nos da un total 3680 muestras de palabras; todas estas adquisiciones han sido procesadas por el sensor y pasadas a archivos de texto. La estructura de estos archivos viene explicada en la sección 4.1.

El conjunto de frases, 274 en total, sigue la misma estructura que las palabras, y éstas son dadas para compararlas con las frases fabricadas artificialmente, y así comparar error de los datos reales con el error de nuestra solución.

[1] https://numpy.org
[2] https://docs.python.org/3/library/argparse.html
[3] https://pypi.org/project/dtaidistance/

A continuación se muestra un ejemplo de matriz de la muestra 4 de la palabra ”catorce". Se ha seleccionado una muestra con pocas filas para poder ver el total del número de filas.

$$
\begin{array}{c}
 \\
frame_0 \\
frame_1 \\
 \\
 \\
 \\
 \\
 \\
frame_{m-1}
\end{array}
\begin{array}{c}
\begin{array}{ccccccc}
pl_{p_x} & pl_{p_y} & pl_{p_z} & \cdots & fr_{4_x} & fr_{4_y} & fr_{4z}
\end{array} \\
\begin{pmatrix}
0{,}003 & 0{,}010 & -0{,}000 & \cdots & -0{,}352 & -0{,}882 & -0{,}175 \\
0{,}008 & 0{,}006 & -0{,}013 & \cdots & -0{,}326 & -0{,}834 & -0{,}217 \\
0{,}004 & 0{,}005 & -0{,}006 & \cdots & -0{,}278 & -0{,}747 & -0{,}289 \\
-0{,}006 & -0{,}018 & 0{,}002 & \cdots & -0{,}205 & -0{,}616 & -0{,}388 \\
0{,}005 & 0{,}011 & -0{,}003 & \cdots & -0{,}112 & -0{,}453 & -0{,}499 \\
0{,}004 & -0{,}008 & 0{,}004 & \cdots & -0{,}009 & -0{,}273 & -0{,}613 \\
0{,}017 & 0{,}010 & -0{,}003 & \cdots & 0{,}093 & -0{,}095 & -0{,}717 \\
0{,}013 & 0{,}005 & 0{,}002 & \cdots & 0{,}181 & 0{,}064 & -0{,}799 \\
-0{,}008 & 0{,}012 & 0{,}018 & \cdots & 0{,}245 & 0{,}184 & -0{,}848
\end{pmatrix}
\end{array}
$$

Donde cada *frame* es un vector de características. Que sigue la forma:

$$
\begin{pmatrix}
ph_{p_x} & ph_{p_y} & ph_{p_z} & ph_{r_x} & ph_{r_y} & ph_{r_z} & fh_{i_x} & fh_{i_y} & fh_{iz}
\end{pmatrix}
$$

Donde p serían componentes de la palma, f los componentes de los dedos. El subíndice p o r haría referencia a la posición y rotación de la palma, respectivamente. La variable $h \in [right, left]$ representaría la mano en cuestión, y $i \in [0, 1, 2, 3, 4]$ cada uno de los dedos de una mano. Todas las palabras y frases, tienen este mismo vector de características, donde $frames \in [0, m-1]$, siendo m el número de filas. Las componentes siguen el orden mostrado, donde primero está la mano izquierda y luego la derecha.

4.3 Arquitectura y diseño detallado

El lenguaje escogido para realizar el proyecto es Python, debido a su flexibilidad de uso y su facilidad a la hora de trabajar con estructuras de datos matriciales.

El sistema está compuesto por módulos (clases). Cada clase corresponde a una de las fases de la solución explicadas anteriormente, junto con otras clases auxiliares que sirven de ayuda para comprender mejor el código y hacer que la solución esté estructurada, de forma que sea más fácil de entender y corregir errores llegado el caso.

Dicho esto, entramos en más detalle en cada clase, con sus funciones y atributos, así como las relaciones entre clases, tal y como podemos observar en el diagrama de clases de la Figura 4.3, donde se muestran los atributos de clase, los métodos y las relaciones entre clases.

A continuación se explica con detalle cada clase.

4.3.1. Clase Launcher

Es la clase donde se inicia la ejecución del programa. Su función es la de manejar la ejecución, recibiendo los parámetros del programa y encargándose del flujo de ejecución según los parámetros de entrada. También es la encargada de mostrar la solución final.

Dispone de las siguientes funciones:

- Función `main` (fuera de la clase): recibe los parámetros de entrada y según estos, lanza a ejecución la función correspondiente de la clase `Launcher`.
- `generateAllWords`: función que lee todas las muestras de palabras y saca las estadísticas del reposo de todo el corpus de palabras, si esa es la opción elegida.
- `generateAllSentencesRandom`: genera todas las frases disponibles, es decir, todas las frases que hay en el corpus de frases originales, utilizando muestras aleatorias de cada palabra para formar la frase.
- `generateAllSentences`: genera todas las frases con las muestras de palabras que indica un fichero de texto previamente escrito, con el número de muestra de cada palabra al formar la frase; esto sirve para realizar las pruebas de error con las mismas muestras.
- `generateAllSentencesEvaluate`: función que lee todas las muestras de frases originales, las compara con su correspondiente frase generada, y calcula la distancia entre ellas y su ruta de alineamiento, para escribirla en un archivo de imagen.
- `launch`: encargada generar cada frase y escribirla en un fichero de salida.
- `launchEvaluate`: encargada de leer las frases originales.
- `updateCalculator`: actualiza los parámetros de la clase que realiza los cálculos del reposo y la interpolación (clase `Calculator`) por los recibidos de entrada.
- `drawDistances`: función auxiliar que escribe la distancia entre palabras y su grá-

fico en un fichero de imagen.

Dispone de los siguientes atributos de clase:

- 2 objetos de la clase `Reader`, cada uno para leer una palabra.
- 1 objeto de la clase `Calculator`, para realizar los cálculos; inicialmente con valor nulo, se inicializa con las 2 matrices de las palabras a interpolar cada vez.
- 1 objeto de la clase `Evaluator`, que sirve de auxiliar para realizar las comparaciones de las frases generadas y las originales.

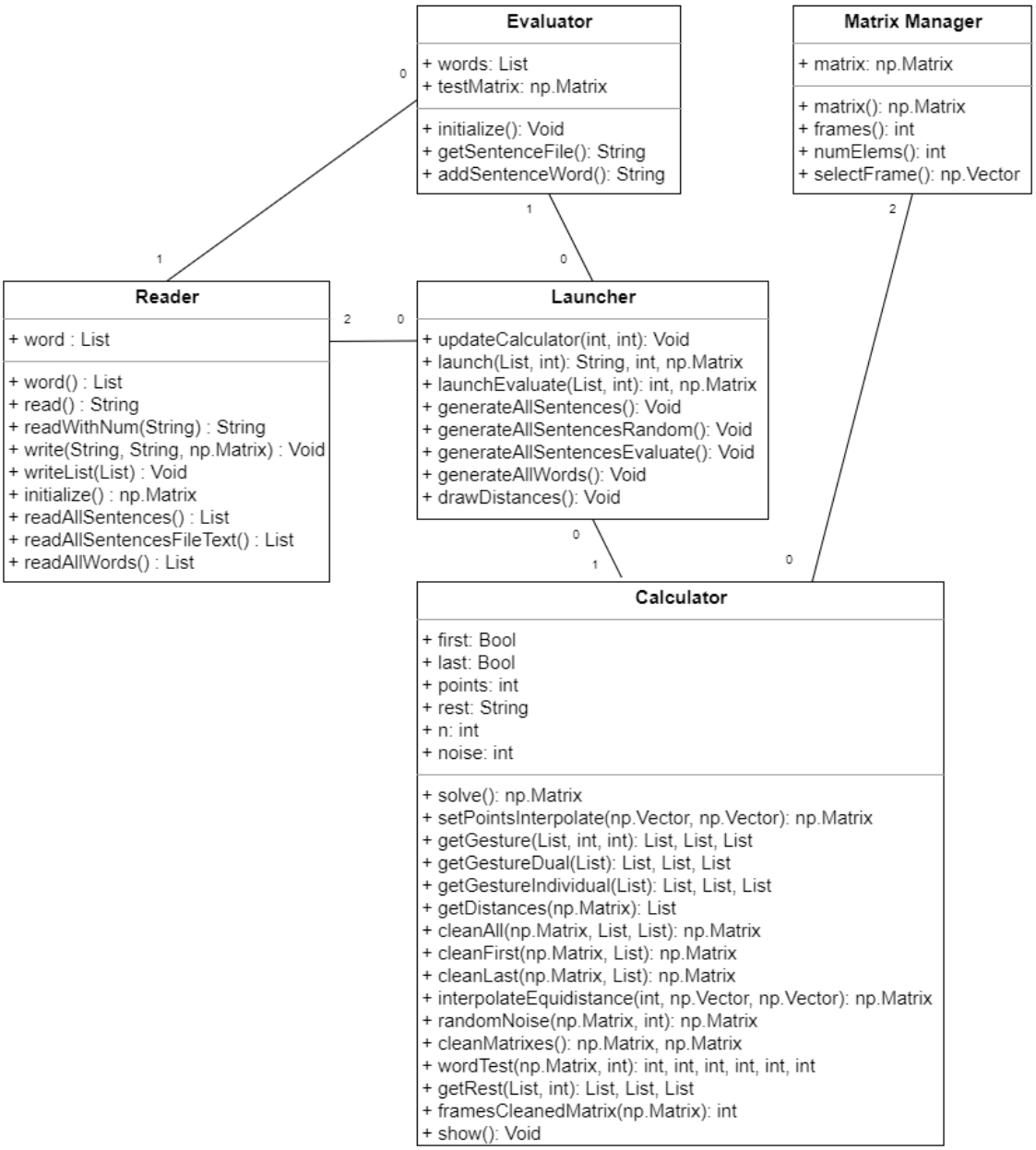

Figura 4.3: Diagrama de clases.

4.3.2. Clase Reader

Se encarga principalmente de la lectura de ficheros, tanto de palabras como de frases generadas, y de escribir también las frases resultantes, así como la inicialización de los ficheros a una estructura matricial dentro de una variable de tipo matriz.

Dispone de las siguientes funciones:

- `read`: se encarga de leer el fichero de la muestra aleatoria de la palabra y pasarla a forma matricial; también se comprueba si el fichero dado existe y en caso negativo, se corta la ejecución del programa.
- `ReadWithNum`: idéntica a la anterior, pero en lugar de leer una muestra aleatoria lee la muestra pasada por parámetros; sirve para realizar las pruebas con las mismas muestras.
- `write`: escribe la matriz en un fichero de texto siguiendo la sintaxis descrita en la sección 4.1.
- `writeList`: escribe las frases generadas y con qué muestras de palabras se generaron en un fichero, para que posteriormente este fichero sea leído para generar las mismas frases de prueba.
- `readAllWords`: lee todas las muestras de palabras y las mete en una lista, para posteriormente ser leídas y generar estadísticas (mediante el uso de `Launcher.generateAllWords`).
- `readAllSentencesFileText`: lee las frases del fichero generado en `writeList` para luego ser leídas (empleando el método `Launcher.generateAllSentences`).
- `readAllSentences`: lee las frases originales para posteriormente ser generadas (usando el método `Launcher.generateAllSentencesRandom`).
- `initialize`: a partir de la lista de números de la palabra, y con un número de columnas conocido (42), saca las dimensiones de la matriz y va recorriendo la lista, inicializando la matriz con el número que corresponda a cada fila y columna.
- Constructor de la clase: inicializa al único atributo, la lista de números que posteriormente se rellena en la función `read`, con valor por defecto nulo; este atributo posee un consultor (*getter*), función donde se devuelve su valor para su uso por clases externas.

4.3.3. Clase Calculator

Su función principal es realizar los cálculos necesarios para obtener el fichero de datos deseado. Es la clase que se encarga de eliminar el reposo y realizar la interpolación entre dos palabras dadas de la frase.

Dispone de las siguientes funciones:

- `wordTest`: la función encargada de sacar las estadísticas del reposo de la palabra, para después acumularlas y dar los resultados globales.
- `solve`: devuelve el resultado final entre dos palabras, para posteriormente ser concatenado con los otros pares de palabras y dar el resultado final.
- `framesCleanedMatrix`: devuelve el número de *frames* de la matriz dada.

- `show`: muestra un punto en un gráfico 3D.

Primera parte, reposo:

- `cleanMatrixes`: se encarga de eliminar el reposo necesario en ambas matrices, con la excepción de eliminar sólo el reposo final en la primera palabra de la frase y sólo el inicial en la palabra final.
- `getDistances`: se encarga de sacar las distancias entre *frames* consecutivos de una matriz.
- `getGesture`: se encarga de separar las distancias de la matriz en reposo inicial, gesto y reposo final dado un umbral de reposo (5, 10, 15 o 20 % del máximo).
- `getGestureDual`: calcula el reposo, variando el umbral en caso de que sea 0, para así tratar de evitar el exceso de reposo (explicado con más detalle en el apartado 5.1).
- `getGestureIndividual`: del mismo modo que el anterior, pero teniendo en cuenta el reposo inicial y final de forma independiente.
- `getRest`: se encarga de llamar a la función de reposo correspondiente según los parámetros de entrada.
- `cleanAll`: elimina ambos reposos de la matriz.
- `cleanFirst`: elimina sólo el reposo final.
- `cleanLast`: elimina sólo el reposo inicial.

Segunda parte, interpolación:

- `interpolateEquidistance`: realiza la interpolación del número de puntos dado y devuelve la matriz correspondiente de los vectores intermedios.
- `setPointsInterpolate`: selecciona el número de puntos a interpolar de forma dinámica, en base a la distancia entre ambas palabras.
- `randomNoise`: genera variación en los datos, modificando sutilmente su valor para generar ruido en los datos interpolados.

Dispone de los siguientes atributos de clase:

- 2 objetos de tipo `MatrixManager`, inicializados con cada matriz con la que se construye el objeto `Calculator`.
- Variable `first` y `last`, que indican si se trata de la primera o última palabra de la frase.
- Variables `points`, `rest` y `n`, el número de puntos a interpolar, el tipo de cálculo del reposo y el porcentaje del mismo, respectivamente.

Todas las variables tienen implementados sus correspondientes métodos modificadores (*setters*).

4.3.4. Clase Evaluator

Se encarga de generar el nombre del fichero de la frase generada, así como de leer los ficheros de las frases originales.

Dispone de las siguientes funciones:

- `initialize`: lee el fichero de la frase original y lo guarda en una variable.
- `addSentenceWord`: añade la siguiente palabra a la lista de palabras de la frase para posteriormente obtener el nombre del fichero de la frase.
- `getSentencesFile`: recoge o genera el nombre del fichero de la frase según la lista de palabras.

Dispone de los siguientes atributos de clase:

- Variable `words`: lista de las palabras que forman la frase.
- Objeto de la clase `Reader`: para leer las matrices.
- Variable `test_matrix`: la matriz leída de la frase original.

4.3.5. Clase MatrixManager

Es una clase auxiliar que proporciona las funciones necesarias para trabajar con la matriz.

Dispone de las siguientes funciones:

- `frames`: devuelve el número de *frames*, es decir, el número de filas.
- `numElems`: calcula el número de elementos de la matriz.
- `selectFrame`: devuelve una fila dada de la matriz.
- `matrix`: devuelve la matriz del objeto `MatrixManager`.

Dispone de los siguientes atributos de clase:

- Una matriz inicializada a partir de la dada en el constructor a la hora de crear el objeto `MatrixManager`.

CAPÍTULO 5
Desarrollo de la solución propuesta

Una vez visto el diseño de la solución, procedemos a explicar el desarrollo más en detalle. En este apartado hablaremos de todas las soluciones que se planearon, las pruebas realizadas, los resultados obtenidos y cómo se obtuvo la solución final. Igualmente, se describen las dificultades encontradas en el desarrollo y cómo se solventaron. También aparecerán fragmentos de código o pseudocódigo, donde se mostrará y explicará la función o algoritmo encargada de realizar una parte crucial de la solución. La parte más importante de la solución se puede subdividir en dos fases: cálculo del reposo de las palabras y cálculo de vectores intermedios entre palabras para generar la frase sintética.

Hablaremos de la entrada del programa, el flujo de datos y la traza.

5.1 Entrada del programa

El programa recoge todos los datos de entrada necesarios para realizar las frases escogiendo todas las opciones disponibles para ello, tales como forma de cálculo del reposo y cantidad de puntos intermedios, entre otras. La Figura 5.1 muestra la ayuda de la aplicación al ejecutar la opción de ayuda en el terminal. A continuación se explica cada uno de los parámetros de entrada y su funcionalidad, completando lo ya visto en el apartado 4.3. Todas las funciones a las que se hace referencia están previamente explicadas en el apartado 4.3.

```
usage: launcher.py [-h] [-v] [-a | -s | -t | -e | -w WORDS [WORDS ...]] [-r {fix,dual,ind}] [-n {5,10,15,20}]
                   [-p {-1,3,4,5,6}] [-d NOISE]

Generation of synthetic phrases in sign language.

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         Increase output verbosity.
  -a, --allWords        Generate repose statistics for all words.
  -s, --allSentences    Generate all sentences.
  -t, --allSentTest     Generate all the sentences from a text file.
  -e, --sentenceEv      Generate stats for all real sentences.
  -w WORDS [WORDS ...]  Generate this phrase.
  -r {fix,dual,ind}     Select the method to calculate the repose: fix, dual or individual. Need parameter -n.
  -n {5,10,15,20}       Number of repose. Fix between 5-20, otherwise 15 or 20.
  -p {-1,3,4,5,6}       Number of points to interpolate (3, 6) or -1 if varible points.
  -d NOISE              Percent of noise aggregate to data.
```

Figura 5.1: Opciones disponibles en el terminal.

Para comenzar la ejecución se necesita al menos uno de estos cinco parámetros, que son mutuamente excluyentes entre ellos:

–allWords: llama a la función generateAllWords.
–allSentences: llama a la función generateAllSentencesRandom.
–allSentTest: llama a la función generateAllSentences.
–sentenceEv: llama a la función generateAllSentencesEvaluate.
-w: recibe las palabras con las que formar la frase y las pasa a una lista con la que posteriormente se lanza la función launch; necesita como mínimo 2 palabras; en caso contrario se produce un error.

Los parámetros opcionales sirven para personalizar el cálculo de las frases variando los parámetros utilizados en su cálculo; no es necesario especificarlos ya que están predefinidos por defecto. Los parámetros son:

-r: permite elegir el tipo de cálculo del reposo entre fijo (fix), dual o individual (ind); el valor por defecto es dual; necesita introducir el parámetro -n.
-n: obligatorio en caso de seleccionar el parámetro -r; en caso contrario, no es utilizado; permite elegir el porcentaje del umbral con el que se calcula el reposo (5, 10, 15 o 20, no se admiten otros valores); el valor por defecto es 10.
-p: permite seleccionar el número de puntos que se quiera generar en la interpolación, entre 3 y 6; en caso de querer que varíen de forma dinámica se usa el valor -1.
-d, permite seleccionar el porcentaje de ruido a agregar a los datos.

Parámetros auxiliares:

–help: muestra la imagen de la Figura 5.1, indicando el uso y explicación de los parámetros.
–verbose: si está activo muestra *logs* de lectura y escritura de ficheros.

La ejecución comienza en el archivo launcher.py, donde se encuentra el método main, llamando a una de las funciones explicadas anteriormente según los parámetros recibidos. Para cada frase se ejecuta el método launch, donde se llama a la clase Calculator para que realice la interpolación de las palabras dos a dos, mediante el método solve. Este método tiene dos comportamientos según se trate o no de la palabra inicial de la frase; en el caso de ser la primera palabra devolvería la matriz de la primera palabra, eliminando su reposo final, concatenada con los puntos intermedios entre las palabras, y la matriz de la segunda palabra, con ambos reposos eliminados. En caso de no ser la palabra inicial se devolvería la matriz de la segunda palabra, eliminando ambos reposos, y concatenada por delante con los puntos intermedios entre la palabra inicial y la devuelta por el método.

Una vez recorridas todas las palabras, launch concatenaría el resultado de cada iteración, para así dar lugar a la matriz final; una vez hecho esto, procederá a escribirla en un fichero .txt con el método write de la clase Reader. Una vez

explicada la traza del programa, a continuación se explica con más detalle los dos problemas a resolver por el mismo.

5.2 Lectura de los datos

En primer lugar, se deben leer los datos de las palabras de los ficheros de texto, y pasarlos a forma de matriz, con la estructura ya conocida (*frames*, columnas=42).

Para ello, se recoge el fichero de la palabra o frase en cuestión y se pasan los datos a una lista, para posteriormente estructurarlos en una matriz mediante el algoritmo 5.1.

Algorithm 5.1 Inicialización de los datos a una matriz

Require: listNums
Ensure: matrix
1: $rows \leftarrow len(listNums)/MAX\_COLS$
2: $row \leftarrow 0$
3: $col \leftarrow -1$
4: $cont \leftarrow -1$
5: **for** $num \in listNums$ **do**
6: **if** $cont < MAX\_COLS$ **then**
7: $cont \leftarrow cont + 1$
8: $col \leftarrow col + 1$
9: **else**
10: $cont \leftarrow 0$
11: $col \leftarrow 0$
12: $row \leftarrow row + 1$
13: **end if**
14: $matrix[row][col] \leftarrow num$
15: **end for**

El coste temporal del algoritmo 5.1 sería $\Theta(n \cdot m)$ debido a que se recorren todos los elementos de la matriz, de n filas y m columnas, pero debido a que m siempre es constante, podemos reducir la fórmula a $\Theta(n)$.

5.3 Cálculo del reposo

En la primera fase de la resolución del problema se trata de detectar el reposo inicial y final de la palabra, en caso de que estuviera claramente definido. Para realizar este proceso, en primer lugar se calcula la distancia euclídea entre los vectores de 42 componentes, es decir, entre cada dos filas consecutivas de la matriz (*frame*), mediante la siguiente fórmula:

$$\begin{aligned} &\forall i \in frames, \\ &\qquad d(i) = \sum_{j=0}^{41} (v[i-1][j] - v[i][j])^2 \end{aligned} \tag{5.1}$$

Siendo v la matriz, e i, j fila y columna, respectivamente. Esto nos da como resultado la distancia entre dos vectores consecutivos de la matriz, es decir, la distancia entre los puntos en un instante de tiempo y los del siguiente instante. Una vez obtenidas estas distancias, se procede a valorar varias técnicas para extraer el estado de reposo. Partiendo de la idea de que en los estados de reposo la mano tiende a estar parada, o realizando movimientos muy despacio, podemos concluir que la distancia entre los puntos en el estado de reposo es muy baja, debido a la lentitud o ausencia del movimiento. Por lo tanto, la idea básica es calcular un cierto umbral, de forma que los *frames* en los que las distancias estén por debajo de éste pertenecerían a las partes consideradas reposo, tanto inicial como final.

Para calcular este umbral podemos servirnos de varias técnicas. En este caso se planteó la técnica del máximo, donde el umbral sería un cierto porcentaje del valor máximo de las distancias de la palabra en cuestión (en el momento en el cual la mano va más rápida), que podía variar entre el 5, 10, 15, o incluso el 20 %. A partir de ese valor, dividimos la palabra en las fases de reposo inicial (*repI*), reposo final (*repF*) y gesto (*gest*). Se observó que no todas las palabras se pueden separar mediante esta técnica, o sólo se puede obtener uno de los dos reposos.

Se muestra en el algoritmo 5.2 cómo se obtienen las partes del reposo a partir de las distancias de la matriz.

Algorithm 5.2 Cálculo del reposo

Require: distances
Ensure: repI, gest, repF
1: $m \leftarrow max(distances)$
2: **for** $i = 0$ **to** $i \leq len(distances)$ **do**
3: $i \leftarrow 0$
4: **while** $i < len(distances) - 1$ **and** *distances*[i] <*threshold*$*m$* **do**
5: $i \leftarrow i + 1$
6: **end while**
7: $j \leftarrow len(distances) - 1$
8: **while** $j > 1$ **and** *distances*[j] <*threshold*$*m$* **do**
9: $j \leftarrow j - 1$
10: **end while**
11: **end for**
12: **for** $k = 0$ **to** k$\leq i$ **do**
13: $repI \leftarrow distances[k]$
14: **end for**
15: **for** $k = i$ **to** k$\leq j + 1$ **do**
16: $gest \leftarrow distances[k]$
17: **end for**
18: **for** $k = j + 1$ **to** k$\leq len(distances)$ **do**
19: $repF \leftarrow distances[k]$
20: **end for**

Respecto al coste temporal del algoritmo 5.2, el primer bucle, tendría un coste $\Theta(n \cdot (i + j))$, siendo n las distancias entre *frames* de la matriz. Las distancias que pertenecen a los *frames* en reposo inicial y final se representan mediante i y j respectivamente. El segundo bucle tendría un coste de $\Theta(i)$, el tercero de $\Theta(j - i)$ y, por último, el cuarto sería de $\Theta(n - j)$. En conclusión, tenemos como resultado final un coste $\Theta(n \cdot (i + j) + n)$, que podríamos simplificar en $\Theta(n \cdot t + n)$, siendo t el total de distancias que pertenecen al reposo.

Una vez dicho esto, se plantearon una serie de pruebas para seleccionar la mejor forma de calcular las distancias que se considerarían en reposo. Lo óptimo sería, por lo tanto, que todas las palabras tuvieran reposo, pero sin excederse, ya que se observó que, utilizando esta técnica, había muchas palabras con excesivamente pocos *frames* considerados como gesto. Para ello se realizaron una serie de pruebas, donde se calculó el reposo de todas las muestras de palabras (un total de 3680) y se estudió las características del reposo de éstas, a fin de hallar la resolución con resultados más equilibrados (es decir, palabras con un número razonable de *frames* en reposo, ni en exceso ni en defecto). Para ello, tomamos como referencia el porcentaje de palabras sin reposo inicial, sin reposo final y sin ambos reposos, así como también el porcentaje de palabras con exceso de reposo inicial, reposo final y ambos reposos. Consideramos como exceso inicial o final aquellas palabras que tengan más del 30 % de los *frames* totales considerados como reposo inicial o final, y exceso total como palabras que tengan más del 60 % del reposo total con respecto al total de la palabra.

A continuación, se detallan las pruebas realizadas y los resultados.

5.3.1. Porcentaje fijo

En primer lugar, se probó con el valor del 5, 10, 15 y 20 % de máximo, y se obtuvieron los resultados de la Tabla 5.1, donde en (*) se presentan los mejores resultados de cada caso.

Tabla 5.1: Porcentaje fijo del máximo.

	Ausencia de reposo			Exceso de reposo			
%Máx	%RepI	%RepF	%RepT	%RepI	%RepF	%RepT	%ErroresT
5	85,4 %	64,7 %	59,6 %	1,3 %*	4,6 %*	17,6 %*	77,2 %
10	75,3 %	52,1 %	44,6 %	2,6 %	8,8 %	29,3 %	73,9 %
15	66,9 %	42,7 %	34,6 %	3,7 %	12,2 %	38,4 %	73,0 %*
20	60,6 %*	35,9 %*	27,7 %*	5,0 %	14,7 %	46,0 %	73,7 %

Como se puede observar, el porcentaje del máximo es inversamente proporcional a la ausencia de reposo, y directamente proporcional al exceso. Esto es algo bastante lógico, debido a que las palabras con un reposo equilibrado calculado con el 5 % se exceden si aumentamos este porcentaje. Por el contrario, si restringimos menos la selección del reposo, aumentando el porcentaje, es posible calcular más reposos de palabras que no lo tenían con un bajo porcentaje.

A raíz de estas conclusiones, se realizaron una serie de pruebas tratando de aprovecharse de las ventajas de ambos casos, para así obtener mejores resultados a nivel de ausencia o exceso total.

5.3.2. Porcentaje variable dual

En este caso, se intentó mejorar el porcentaje total tanto de reposo como de exceso, modificando sólo el porcentaje del máximo en caso de que ambos reposos fueran 0, y así tratar de no "estropear" los resultados donde se obtenía correctamente el reposo a un bajo porcentaje, a fin de no excederlo. Los resultados obtenidos fueron los mostrados en la Tabla 5.2.

Tabla 5.2: Porcentaje variable dual del máximo.

	Ausencia de reposo			Exceso de reposo			
%Máx	%RepI	%RepF	%RepT	%RepI	%RepF	%RepT	%ErroresT
5 - 15	75,5 %	46,3 %	34,6 %	1,6 %*	5,1 %*	21,8 %*	56,4 %
5 - 20	72,9 %*	41,3 %*	27,7 %*	1,7 %	5,2 %	22,6 %	50,3 %*

En este caso, se observa que los resultados mejoran significativamente en todos los aspectos respecto al experimento anterior, excepto el caso de ausencia del reposo inicial y ligeramente el final. Sin embargo, a nivel global son superiores

al caso anterior. Observamos que el hecho de incluir el 20 % mejora muy ligeramente el caso de la ausencia de reposo, a costa de empeorar muy ligeramente el exceso, por lo que consideramos que vale la pena incluir este 20 %.

Se muestra en el algoritmo 5.3 el pseudocódigo del algoritmo que recalcula el reposo en caso de que ambos reposos sean nulos.

Algorithm 5.3 Cálculo del reposo dual

Require: distances
Ensure: repI, gest, repF
1: $repI, gest, repF \leftarrow getGesture(distances, 5)$
2: **if** $len(repI) = 0$ and $len(repF) = 0$ **then**
3: $repI, gest, repF \leftarrow recalculateGesture(distances, 10)$
4: **if** $len(repI) = 0$ and $len(repF) = 0$ **then**
5: $repI, gest, repF \leftarrow recalculateGesture(distances, 15)$
6: **if** $len(repI) = 0$ and $len(repF) = 0$ **then**
7: $repI, gest, repF \leftarrow recalculateGesture(distances, 20)$
8: **end if**
9: **end if**
10: **end if**

Respecto al coste del algoritmo 5.3, al realizar llamadas al algoritmo 5.2, el coste sería igual al del algoritmo 5.2, $\Theta(n \cdot t + n)$.

5.3.3. Porcentaje variable inicial y final

Debido a los resultados anteriores, se observa que con el variable dual se obtienen muy buenos resultados a nivel tanto de ausencia como de exceso global, pero no tanto a nivel individual de reposo inicial o final, por lo que se realizaron experimentos teniendo en cuenta también los casos en los que sólo exista uno de estos dos reposos y se trate de mejorar ese, sin estropear el que sí que funciona correctamente.

En este experimento se comprobó en qué casos el reposo individual y total era 0, y se recalculó variando el porcentaje únicamente en el caso es necesario (inicial, final o ambos). Se obtuvieron los resultados mostrados en la Tabla 5.3.

Tabla 5.3: Porcentaje variable individual del máximo.

	Ausencia de reposo			Exceso de reposo			
%Máx	%RepI	%RepF	%RepT	%RepI	%RepF	%RepT	%ErroresT
5 - 15	66,9 %	42,7 %	34,6 %	1,9 %*	5,2 %*	25,6 %*	60,2 %
5 - 20	60,6 %*	35,9 %*	27,7 %*	2,2 %	5,3 %	28,2 %	55,9 %*

En este caso, se observa que al tener en cuenta los casos donde sólo existe un reposo, ya sea inicial o final, mejora significativamente la ausencia del reposo individual, sin empeorar el global. Por otro lado, en el caso de utilizar el 20 %, sí que empeora ligeramente los resultados anteriores del exceso de reposo, sobre todo en el exceso total, donde es un empeoramiento a tener en cuenta.

En el algoritmo 5.4 se muestra el algoritmo que recalcula el reposo en caso de que uno de los reposos sean nulos (reposo variable individual).

Algorithm 5.4 Cálculo del reposo individual

Require: distances, $limit \in [5, 10, 15, 20]$
Ensure: repI, gest, repF
1: $ini, fin \leftarrow 5$
2: **while** $fin \leq limit$ **and** $ini \leq limit$ **do**
3: $repI, gest, repF \leftarrow getGesture(distances, ini, fin)$
4: **if** $len(repI) = 0$ **and** $len(repF) = 0$ **then**
5: $ini \leftarrow ini + 5$
6: $fin \leftarrow fin + 5$
7: **continue**
8: **end if**
9: **if** $len(repI) = 0$ **and** $len(repF)! = 0$ **then**
10: $ini \leftarrow ini + 5$
11: **continue**
12: **end if**
13: **if** $len(repI)! = 0$ **and** $len(repF) = 0$ **then**
14: $fin \leftarrow fin + 5$
15: **continue**
16: **end if**
17: **if** $len(repI)! = 0$ **and** $len(repF)! = 0$ **then**
18: $ini \leftarrow ini + 5$
19: $fin \leftarrow fin + 5$
20: **break**
21: **end if**
22: **end while**

En cuanto al coste temporal del algoritmo 5.4, sería el mismo caso que el comentado en el algoritmo 5.3, un coste $\Theta(n \cdot t + n)$.

5.3.4. Comparación de resultados y conclusión

A la vista de los resultados presentados, realizamos una comparación del porcentaje total de datos que se considerarían incorrectos, es decir, en defecto o en exceso de reposo, para valorar cuál es la mejor opción a la hora de seleccionar el cálculo del reposo.

Tabla 5.4: Comparación porcentajes totales de error para cada técnica del cálculo del reposo.

Técnica	%Ausencia Total	%Exceso Total	%Errores Total
Fijo 5 %	59,6 %	17,6 %*	77,2 %
Fijo 10 %	44,6 %	29,3 %	73,9 %
Fijo 15 %	34,6 %	38,4 %	73,0 %
Fijo 20 %	27,7 %*	46,0 %	73,7 %
Variable Dual 15 %	34,6 %	21,8 %	56,4 %
Variable Dual 20 %	27,7 %*	22,6 %	50,3 %*
Variable Ind. 15 %	34,6 %	25,6 %	60,2 %
Variable Ind. 20 %	27,7 %*	28,2 %	55,9 %

Tal y como se observa en la Tabla 5.4, basándonos en la mejora a nivel global, es decir, que ambos reposos no sean nulos o que no excedan, se ve que el mejor resultado lo proporciona la técnica del porcentaje variable dual de hasta el 20 %, seguido de porcentaje variable individual del 20 %, que sí proporciona mejores resultados a nivel individual, pero ligeramente peores a nivel total.

En conclusión, los mejores resultados totales los proporciona el porcentaje variable, pero en este caso utilizaremos la técnica individual, ya que resulta más completa al reducir significativamente la ausencia del reposo inicial, que es la característica donde más ausencia se genera.

Una vez obtenidos los estados de reposo, simplemente se eliminan en los puntos de unión de las palabras, al final de la primera y al inicio de la segunda, y las palabras quedan sin reposo entre ambas. En caso de no ser posible obtener la información del reposo, la palabra se queda como estaba originalmente.

Respecto al coste temporal de cada una de las técnicas, debido a que el cálculo del reposo dual e individual depende del cálculo fijo (algoritmo 5.2), se podría considerar que todas las técnicas son equivalentes en cuanto a coste temporal, o que el coste fijo es ligeramente mejor en el peor de los casos respecto a los otros dos.

5.4 Cálculo de los vectores intermedios

Una vez hemos limpiado las palabras, se procede a la segunda parte del desarrollo de la solución: calcular los datos intermedios entre dos palabras, para así generar la frase artificialmente.

Para esta segunda parte, en primer lugar debemos decidir la cantidad de puntos intermedios que debemos interpolar. Para ello, primero se estudiaron las distancias entre *frames* de las muestras de frases reales, para poder estimar cómo serían las distancias dentro de la misma palabra y entre palabras. Se observó que entre palabras las distancias tienen una media de 0,13, con valor mínimo de 0 y un valor máximo de 22,5 (en casos anómalos). Esto nos indica que la distancia dentro de la misma palabra generalmente suele estar en el intervalo [0, 0,2], aunque ocasionalmente puede subir hasta valores más altos.

Por otra parte, los valores entre palabras sí que se observaron muy altos, en el intervalo [0, 38,18], con una media de 8,35. Por tanto, lo deseable en este caso sería conseguir unos valores parecidos en las frases generadas sintéticamente, es decir, no excederse mucho en la distancia entre palabras. La distancias dentro de la palabra se consideran las mismas, dado que construimos la frase a partir de éstas.

Se realizó esta misma prueba con las frases sintéticas, para observar cómo variaban estas distancias según variamos en número de puntos. Para ello, en primer lugar, se decidió que los puntos variarían dentro del intervalo de 3 a 6 puntos, tratando de aproximar la distancia media entre palabras lo máximo posible a la realidad. Se observó que, a mayor número de puntos interpolados, aumentaba la diferencia con las estadísticas de las frases reales, por lo que no era conveniente excederse con el número de puntos intermedios entre cada palabra.

Siguiendo la estructura observada en las frases de muestra y observando los resultados, este número debería variar según las características de la palabra, como la distancia entre el punto inicial y final entre los cuales se interpola, para así dar la máxima verosimilitud a la frase formada.

Una vez hecho esto, existen múltiples formas de generar estos puntos. Por ejemplo, se puede generar una ruta lineal entre el punto inicial y final que une las dos palabras, y de la ruta se escogen los puntos intermedios; esta técnica es conocida como segmentación de traza, tal y como se utiliza en [6].

A raíz de esto, surgen nuevas posibilidades, ya que se debe decidir qué puntos de la ruta escoger. La posibilidad que se planteó inicialmente es una de las más sencillas: escoger puntos equidistantes. Una vez escogidos los puntos, es posible que el resultado quede un poco forzado o poco realista, ya que en la realidad no se realizan movimientos siguiendo una ruta lineal de manera perfecta. Por tanto, se realizó un pequeño desplazamiento de los puntos, en uno o más ejes, para dar una mayor sensación de realismo.

En el algoritmo 5.5 se muestra el algoritmo que introduce el ruido en los datos.

Algorithm 5.5 Cálculo del ruido en los datos

Require: matrix, percent
Ensure: matrixWithNoise
1: **for** $frame = 0$ **to** $frame \leq lastFrame$ **do**
2: **for** $col = 0$ **to** $col \leq MAX\_COLS$ **do**
3: $max \leftarrow matrix[frame][col] * percent$
4: $noise \leftarrow random(0, |max|)$
5: **if** $random(true, false)$ **then**
6: $matrix[frame][col] \leftarrow matrix[frame][col] + noise$
7: **else**
8: $matrix[frame][col] \leftarrow matrix[frame][col] - noise$
9: **end if**
10: **end for**
11: **end for**

El coste temporal del algoritmo 5.5 es $\Theta(n \cdot m)$, siendo n el número de filas y m el número de columnas de la matriz, respectivamente.

CAPÍTULO 6
Pruebas

En este apartado se muestran las pruebas realizadas para la validación del trabajo, así como los resultados obtenidos en ellas, y lo que supusieron a la hora de presentar la solución final del trabajo.
Para la validación de los resultados finales se realizó el reconocimiento por HMM de las palabras generadas artificialmente, para así calcular su error de reconocimiento y poder ver si la solución es factible.
Se realizó otra prueba auxiliar orientativa, para ayudar a comprender mejor la naturaleza de las frases generadas y poder compararlas con las reales. Dichas pruebas se detallan a continuación.

6.1 Reconocimiento de las frases sintéticas con HMM

La principal prueba que permite descubrir si las frases sintéticas son equivalentes o semejantes a las reales es realizando un reconocimiento de las frases generadas por un modelo entrenado de HMM, para comprobar si tiene un error de reconocimiento aceptable y semejante al de las frases reales disponibles. De esa manera se podrían considerar realistas y utilizarse de manera efectiva tal como las reales.

Para realizar esta prueba se recreó el mismo escenario que se utiliza para reconocer frases reales, y así sacar el error de reconocimiento de las frases artificiales como si de las reales se tratase. Se hicieron las pruebas sobre las frases generadas artificialmente generadas de maneras distintas, cambiando entre los varios tipos de reposo y el número de puntos entre palabras. Las frases de prueba se generaron con las mismas muestras de palabras, es decir, frases formadas por la unión de muestras aleatorias de cada palabra, y utilizando las mismas muestras para cada prueba, para que los resultados sean comparables entre sí. Se realizaron las pruebas por cada tipo de reposo explicado en el apartado 5.3, y por cada número de puntos a interpolar.

En algunos casos el error de reconocimiento no varía de un caso a otro, por lo que en los resultados se muestran sólo los cambios significativos.

Los resultados obtenidos fueron los presentados por la Tabla 6.1.

Tabla 6.1: Error de reconocimiento con puntos fijos.

	% Error			
Técnica	3 puntos	4 puntos	5 puntos	6 puntos
Fijo	6,6 %	6,6 %	6,6 %	6,7 %
15 % Dual	6,6 %	6,7 %	6,7 %	6,8 %
20 % Dual	6,6 %	6,8 %	6,8 %	6,9 %
Individual	6,6 %	6,7 %	6,7 %	6,8 %

Como se puede observar, el porcentaje de error en el reconocimiento es muy similar en todas las pruebas realizadas, y apenas varía de un caso a otro. La conclusión que se saca de estos resultados es que, a mayor número de puntos, mayor error en todos los casos. También se observa que los porcentajes fijos dan lugar a menos error.

La Tabla 6.2 muestra los resultados con un número variable de puntos según la distancia entre las dos palabras.

Tabla 6.2: Error de reconocimiento con puntos variables.

Técnica	%Error
Fijo 5 %	6,5 %
Fijo 10 %	6,6 %
Fijo 15 %	6,6 %
Fijo 20 %	6,5 %

Se puede observar que son muy semejantes a los resultados anteriores, aunque mejoran ligeramente el error.

En conclusión, los resultados obtenidos son muy buenos, con una tasa de error general bastante baja (6,5 % aproximadamente) frente al reconocimiento de las frases reales, un 16 % aproximadamente. Por lo tanto, esto nos indica que los resultados generados artificialmente son demasiado buenos en comparación con la realidad; por lo tanto, no llegan a ser una buena representación de la misma. Por este motivo se decidió empeorar estos resultados un poco, para acercarlos más a la realidad y a un porcentaje de error del 16 %.

Uno de los posibles factores que hicieron que estos resultados sean mejores a los reales puede ser el hecho de que los puntos interpolados se generen en una ruta lineal entre los dos vectores inicial y final de las palabras, por lo que eso resulta un poco forzado y no representa del todo bien el movimiento real de las manos. Para corregir esto, como se comentó previamente en el apartado 5.4, se deberá meter "ruido" y una desviación de la ruta lineal que une a ambos vectores, para aumentar así el error de reconocimiento.

En un primer momento, se probó a meter ruido únicamente en los datos interpolados, pero se observó que prácticamente no variaban los resultados, debido

que la cantidad de datos interpolados es ínfima frente al total de los datos. Por este motivo se decidió introducir el ruido en todas las filas de la matriz, es decir, tanto en las características de las palabras originales que se concatenaban como en los vectores interpolados que las unían. Se obtuvieron los resultados presentados en la Tabla 6.3

Tabla 6.3: Error de reconocimiento con ruido en todos los datos.

Ruido en los datos	%Error
20 %	7,0 %
30 %	8,2 %
40 %	11,5 %
50 %	14,7 %
55 %	16,6 %
60 %	20,3 %
65 %	21,8 %
70 %	24,5 %

Tal y como se observa, se ha conseguido aumentar significativamente el error de reconocimento al introducir ruido en todos los datos, y no sólo en los datos interpolados, lo que era de esperar. En el caso de introducir un 55 % del ruido, se obtiene un error de reconocimiento del 16,6 %, con intervalo de confianza de +-2,7, frente a los datos originales, con error de 16,3+-2,4. Se puede observar que ambos intervalos se solapan, el sintético de con un intervalo de error [13,9, 19,3], frente al original en el intervalo [13,9, 18,7].

En esta ocasión, para realizar las pruebas, a pesar de que todas las pruebas realizadas tenía un error muy similar, se escogió la generación de las frases sintéticas con reposo fijo y el umbral del 10 %, debido a que en el apartado anterior se observó que eran las que proporcionaban menor error, a pesar de que en esta ocasión se buscada aumentarlo.

Vistos los resultados, y dado que se ha obtenido el resultado deseado (generar frases sintéticas que tengan la máxima semejanza con las originales y que puedan ser equivalentes debido al mismo margen de error de reconocimiento), se puede concluir que el trabajo ha sido exitoso y se obtuvieron los resultados deseados.

6.2 Distancias con las muestras de frases originales

Como técnica auxiliar, se utiliza la técnica de *Dynamic Time Warping* (DTW) para calcular la similitud o la distancia entre la frase original y la frase generada artificialmente. Para ello, se utilizó la librería de Python que se explica con más detalle en el apartado 4.1. Para comparar ambas frases, nos basamos en la distancia calculada mediante DTW y en la gráfica resultante del alineamiento entre las dos matrices. Un ejemplo de la representación de la frase ”el niño dos" se pueden observar en la Figura 6.1.

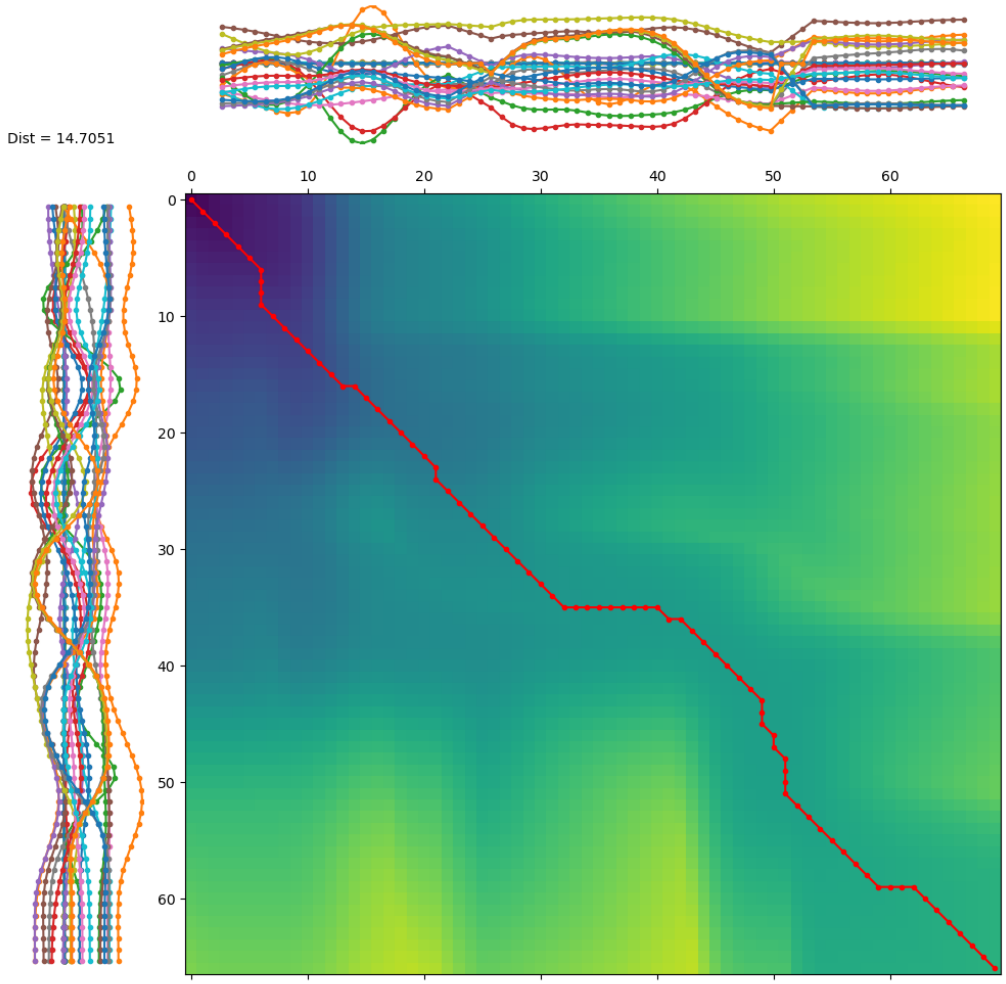

Figura 6.1: Resultados obtenidos de aplicar DTW entre la frase original y la sintética de ”el niño dos" .

En la parte superior se puede ver la representación de la frase generada y en la parte izquierda la de la frase original. Cada línea corresponde a un vector columna; los puntos de cada línea representarían la fila correspondiente, y el valor de la columna (línea) y la fila (punto) sería el valor correspondiente a ese elemento de la matriz[punto][línea]. En la representación no se muestran todas las columnas debido a su gran número.

En el recuadro central podemos observar que el eje X corresponde al número

de filas de la frase generada; de la misma forma, el eje Y corresponde a la frase original. Los recuadros coloreados representan el valor que tomaría la distancia entre los dos *frames* de ambas frases que se correspondan en ese punto. El color representa gráficamente la distancia: a color más oscuro, menor distancia. La línea roja representa el mejor alineamiento entre las dos matrices.

En la esquina superior izquierda se observa la distancia mínima entre ambas frases. Es importante recalcar que en la Figura 6.1 no se aplicó ningún tipo de preproceso o normalización para calcular la distancia y la ruta de alineamiento.

En esta sección se ha explicado el funcionamiento de la ruta de alineamiento y cómo se muestra. A continuación se explicarán los pasos seguidos para realizar las pruebas.

6.2.1. Normalización de los datos

Lo más habitual a la hora de alinear dos series de tiempo mediante la técnica DTW es aplicar un preproceso a los datos, ya sea una normalización o estandarización, tal y como se muestra en [9], donde se comparó aplicar DTW a datos con o sin normalizar.

En el caso de este trabajo, se decidió que el paso previo a calcular la distancia entre las dos frases mediante el algoritmo DTW sería normalizar los datos, debido a que puede haber diferencias significativas en caso de que las frases tengan unas dimensiones muy distintas en su escala de valores. Para que el algoritmo se centre en las diferencias estructurales, las matrices se normalizaron antes de calcular la distancia aplicando la Z-normalización. La ecuación (6.1) muestra la función que calcula los nuevos valores de cada elemento.

$$x'_{i,j} = \frac{x_{i,j} - \mu_i}{\sigma_i} \tag{6.1}$$

Donde i es la fila, j la columna, μ_i y σ_i son la media y la desviación de típica de la fila i, respectivamente; este proceso se realizó mediante la librería scipy[1]. Esta normalización se haría en ambas matrices y, una vez hecho esto, se calcularía la distancia DTW.

En la Figura 6.2 se muestra la ruta de alineamiento de la frase anterior aplicando la Z-normalización.

[1] https://docs.scipy.org/doc/

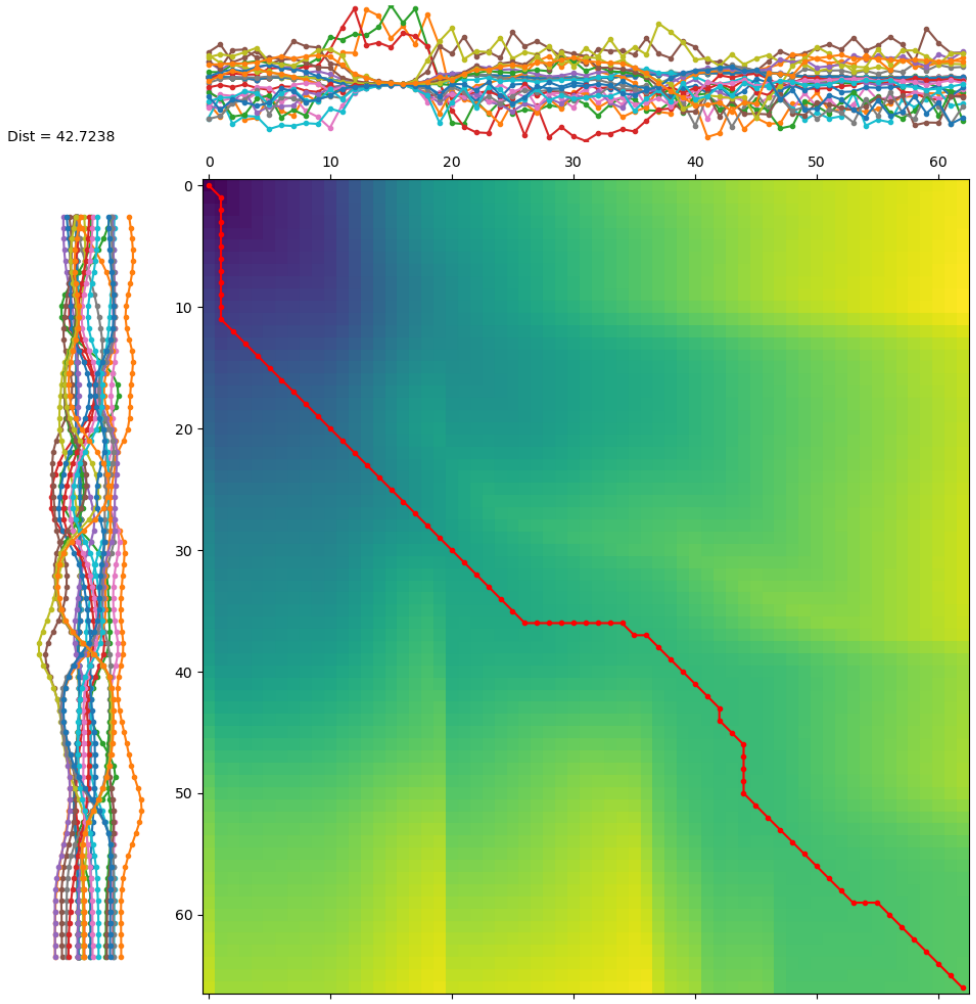

Figura 6.2: Resultados obtenidos de aplicar DTW entre la frase original y la sintética de ”el niño dos" aplicando normalización.

Como se puede observar, los vectores de la Figura 6.2 están más solapados entre sí debido a la normalización. No obstante, se puede observar un pico en algunos de los vectores de la frase sintética; esto puede ser debido a que en algunas ocasiones la normalización puede introducir sesgos en el caso de que la varianza del vector sea muy pequeña, en cuyo caso la propia normalización aumentaría el ruido.

En la Tabla 6.4 se muestran los resultados de calcular las distancias entre frases, con o sin normalizar, con un determinado ruido.

Tabla 6.4: Distancias DTW con o sin normalización.

	Distancias sin normalizar			Distancias con Z-Normalización		
%Ruido	Máxima	Mínima	Media	Máxima	Mínima	Media
20 %	88,2	14,2	37,7	131,4	30,0	64,8
30 %	88,6	14,4	37,9	126,8	29,5	63,4
40 %	87,5	14,8	38,3	127,7	30,1	63,8
50 %	87,4	15,2	38,8	126,2	31,7	64,2
55 %	88,4	15,6	39,2	126,5	33,1	64,6
60 %	88,2	15,7	39,4	126,1	34,3	65,0
65 %	87,8	15,8	39,8	126,1	34,0	65,3
70 %	89,4	16,7	40,3	126,0	34,8	65,6

Como se observa, los resultados empeoran al aplicar Z-normalización, debido a que aumenta la distancia. Se realizó también el cálculo de la distancia dividiendo por la máxima longitud de las palabras a comparar (original y sintética) para obtener unos resultados normalizados por longitud. Los resultados de observan en la Tabla 6.5.

Tabla 6.5: Distancias DTW con o sin normalización y normalizando por longitud.

	Distancias sin normalizar			Distancias con Z-Normalización		
%Ruido	Máxima	Mínima	Media	Máxima	Mínima	Media
20 %	0,54	0,11	0,24	0,75	0,22	0,40
30 %	0,55	0,11	0,24	0,75	0,22	0,40
40 %	0,54	0,11	0,24	0,76	0,22	0,40
50 %	0,54	0,11	0,25	0,76	0,23	0,41
55 %	0,55	0,12	0,25	0,76	0,23	0,41
60 %	0,55	0,12	0,25	0,76	0,24	0,41
65 %	0,57	0,12	0,25	0,77	0,24	0,42
70 %	0,56	0,12	0,26	0,77	0,24	0,42

En conclusión, se puede afirmar que los resultados aplicando la Z-normalización son peores que sin aplicarla. Esto puede ser debido a que no sea necesario aplicar dicha normalización, debido a que los rangos de valores en ambas matrices son similares, y la normalización ha empeorado los resultados. Por otra parte, también se observa que la distancia es directamente proporcional al ruido aplicado a los datos a pesar de su escasa influencia, como era de esperar.

CAPÍTULO 7
Conclusiones

El objetivo del proyecto era realizar una generación de frases en lengua de signos a partir de palabras sueltas, lo que permitiría utilizar esas frases sintéticas de la misma forma que las reales.

Eso tiene como objetivo conseguir una forma mucho más rápida, fácil y factible de obtener una cantidad masiva de frases, para así poder mejorar notablemente cualquier tipo de sistema de reconocimiento de frases en lengua de signos. Esto evitaría la gran dificultad de obtener estas frases mediante un sensor de la misma manera que se obtienen las palabras, debido a la amplia combinación de palabras que hay a la hora de realizar frases.

En este trabajo se ha conseguido generar un conjunto de 274 frases a partir de un vocabulario de 92 palabras. Se podían haber generado todas las posibles frases con estas 92 palabras, lo que implica que, tan sólo aumentando el vocabulario, se pueda llegar a obtener una cantidad casi ilimitada de frases que consigan complementar a las originales, sin necesidad de obtenerlas mediante un sensor. Esto llevaría a una gran mejora en los sistemas de reconocimiento y la posibilidad de mejorar este tipo de sistemas que implantan reconocimiento de lengua de signos.

Dicho esto, y tal como se indica en las pruebas, podemos afirmar que hemos cumplido los objetivos del proyecto, ya que hemos conseguido resultados muy similares a los datos originales.

Uno de los problemas encontrados en este trabajo es referente a la obtención de las partes de reposo de las palabras, debido a que no hay ninguna información o referencia a ellas para poder comparar o asegurar que esté correctamente detectado. Debido a esto se utilizó la técnica de la distancia, como manera más lógica de obtenerlo, y debido a la incapacidad de corroborar la técnica no se pudo ajustar perfectamente este cálculo, aunque sí se pudo aproximar. Por otro lado, debido a que el reposo extraído es bajo frente al resto de los datos, esto no afecta demasiado al desarrollo de la solución, ya que la parte importante de los datos reside en las palabras que forman la frase.

En conclusión, tal y como indican las pruebas realizadas, se ha conseguido el objetivo final del proyecto, y se han obtenido los resultados deseados.

7.1 Relación con los estudios cursados

Como estudiante de Ingeniería Informática de la rama de Computación, escogí este TFG debido a el tema que trataba y las técnicas que se utilizan para realizar el proyecto, debido a mis conocimientos en este ámbito.

Durante el proyecto utilicé el lenguaje de programación Python, aprendido durante la rama por su gran utilidad en todo tipo de algoritmos, y especialmente en su trato con matrices y su sencillez de uso.

A nivel conceptual, durante el proceso de realización del proyecto consolidé conceptos ya conocidos durante la rama relacionados con la Inteligencia Artificial y las técnicas de reconocimiento, debido al estudio que realicé para el capítulo 2. Esto también sirvió para seleccionar técnicas adecuadas para la realización de las partes cruciales del TFG, como indico a lo largo del trabajo. Esto me llevó a aprender nuevos conceptos, como DTW, entre otros, además de conocer qué técnicas son las más utilizadas en el ámbito del reconocimiento de lengua de signos y el contexto en el que se mueve.

A nivel técnico utilicé los conocimientos sobre matrices y algoritmos vistos en la asignatura de cuarto curso, Algorítmica, así como los conocimientos sobre el lenguaje de programación que aprendí en la asignatura de Sistemas del almacenamiento y recuperación de información. También aprendí cosas nuevas a nivel de programación con matrices.

En conclusión, el proyecto me sirvió tanto para consolidar y ampliar conceptos conocidos en la carrera, como para conocer y descubrir nuevos conceptos y técnicas. Además, la redacción del mismo con el procesador de texto LaTeX me permitió iniciarme en el uso de esta herramienta para el desarrollo de documentación.

CAPÍTULO 8
Trabajos futuros

Los resultados de este trabajo y su desarrollo permiten una gran variedad de avances en cuanto a todo tipo de trabajos relacionados con la lengua de signos. El hecho de poder generar cualquier frase a partir de palabras aisladas permite, principalmente, la generación de una cantidad masiva de datos de frases. Esto abre un mundo de posibilidades; principalmente serviría para el entrenamiento de sistemas de reconocimiento de lengua de signos, así como para el entrenamiento de modelos más potentes, como, por ejemplo, redes neuronales profundas [16]. La ventaja de esto es que sólo sería necesaria la adquisición de un amplio vocabulario de palabras (más allá de las 92 utilizadas en este trabajo), y así se podría generar tal cantidad de datos, lo que permitiría mejorar el error de reconocimiento.

La mejora de este error supondría la creación o mejora de sistemas en los que se utilice el reconocimiento de lengua de signos, así como ser capaz de implementarlo eficientemente en otras aplicaciones ya existentes (como por ejemplo, Hand Talk [1], cuya imagen se muestra en la Figura 8.1), para así poder hacer que sean accesibles a personas con problemas auditivos o en el habla. Es el caso de esta herramienta, que permite tanto reconocer gestos del usuario como recrearlos si fuera necesario, para así tener una comunicación más fluida con estas personas.

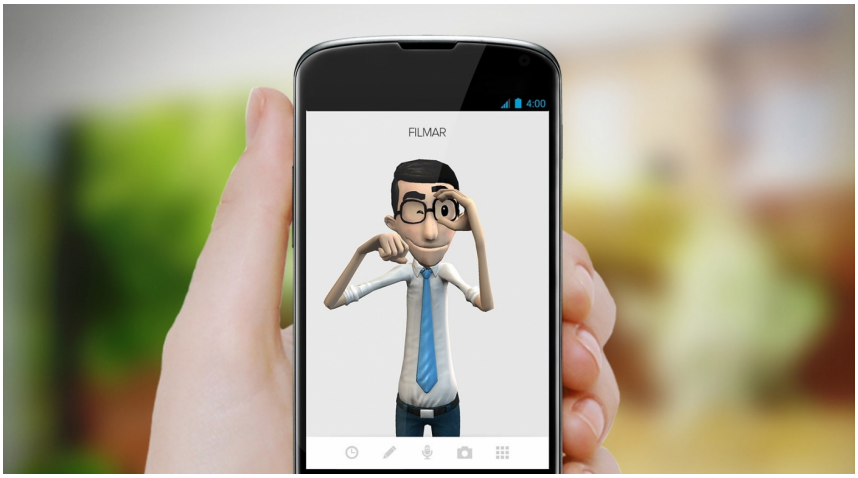

Figura 8.1: Aplicación Hand Talk, intérprete de la Lengua de Signos Portuguesa.

[1] https://www.handtalk.me/br

Un posible trabajo futuro sería ese, adquirir más muestras de palabras y, una vez hecho esto, realizar un sistema de generación de frases a partir de un vocabulario, en el cual el sistema sería capaz de generar frases aleatorias con sentido a partir de un vocabulario dado, o incluso generar todas las posibles frases formadas con estas palabras.

También se puede aprovechar la parte de desarrollo del trabajo para otros trabajos que tengan la misma estructura o parecida (datos en forma de matriz, más o menos estructurada) y, a partir de éstas, generar nuevas matrices como se realiza en este trabajo. En caso de no tener tal estructura, se pasarían los datos a una forma matricial, con la que se podría trabajar de esta manera. Por ejemplo, composición de canciones (codificadas en matrices), donde, dadas dos o más canciones, se crearía una composición a partir de éstas (fusionarlas, concatenarlas, intercalarlas).

Otra de las posibles aplicaciones del trabajo sería en el ámbito de la enseñanza; a partir de la generación de frases desde palabras, se podrían crear sistemas de enseñanza de lengua de signos donde el alumno aprendería a crear frases, y el sistema puntuaría al alumno en base a su ejecución del gesto, o mostrándole la frase en cuestión como ejemplo.

APÉNDICE A
Glosario

A.1 Red Neuronal (NN)

Una red neuronal artificial es un grupo interconectado de nodos similar a la red de neuronas en un cerebro biológico. Consiste en un conjunto de unidades, llamadas neuronas artificiales, conectadas entre sí para transmitirse señales. Su objetivo es resolver los problemas de la misma manera que el cerebro humano. La información de entrada atraviesa la red neuronal (donde se somete a diversas operaciones) produciendo unos valores de salida. [1]

A.2 Mixturas Gaussianas (GMM)

Un modelo de mixturas gaussiana (GMM) es una función de densidad de probabilidad paramétrica representada como una suma ponderada de densidades de probabilidad gaussianas. Los GMM se usan comúnmente como un modelo paramétrico de la distribución de probabilidad de características continuas en sistemas de reconocimiento de voz o similares. Los parámetros del GMM se estiman a partir de los datos de entrenamiento utilizando el algoritmo iterativo Esperanza-Maximización (EM) o Máxima Estimación posteriori (MAP) de un modelo previo bien entrenado.

La idea básica del algoritmo EM es, comenzando con un modelo inicial λ, estimar un nuevo modelo λ', de modo que la probabilidad de X (vector de entrenamiento) del nuevo modelo sea mayor que para el modelo inicial. El nuevo modelo se convierte en el modelo inicial para la próxima iteración y el proceso se repite hasta se alcanza cierto umbral de convergencia. En cada iteración EM, se utilizan fórmulas de reestimación que garantizan un aumento monótono de la verosimilitud de las muestras de entrenamiento para el modelo.

En el caso de MAP, para la adaptación estas ”nuevas” estimaciones estadísticas se combinan las estadísticas ”viejas” de los parámetros de mezcla anteriores utilizando un coeficiente de mezcla dependiente de datos [15].

[1] https://es.wikipedia.org/wiki/Red_neuronal_artificial

A.3 Modelos Ocultos de Markov (HMM)

Un modelo oculto de Markov es un autómata de estados finitos que permite modelar procesos estadísticos donde la ocurrencia de los estados está asociada con una distribución de probabilidad y las transiciones entre estados funcionan mediante un conjunto de probabilidades (probabilidades de transición entre estados). La ocurrencia de cada estado depende del estado en el instante anterior [11].

A.4 K-vecinos más cercanos (KNN)

El método de los k vecinos más cercanos es un método de vecindad basado en casos o instancias. En el caso de k-nn, cada vez que se va a clasificar un dato, en la fase de entrenamiento se elabora un modelo específico para cada nuevo dato, y una vez que éste se clasifica sirve como un nuevo caso de entrenamiento para clasificar una nueva instancia. Para clasificar un nuevo dato, se ubica el dato a clasificar en el espacio de representación y se determina un "radio de vecindad"; después se traza una circunferencia cuyo centro es el dato a clasificar; se determina el valor de k, se asigna la clase al nuevo elemento de acuerdo al valor de k y al número de datos encerrados en la circunferencia [13].

A.5 *Dynamic Time Warping* (DTW)

La técnica DTW se utiliza en el análisis de series temporales. Se trata de un algoritmo para medir la similitud entre dos secuencias temporales, que pueden variar en velocidad. Además de una medida de similitud entre las dos secuencias, se puede obtener una llamada "ruta de alineación", al alinear de acuerdo con esta ruta las dos secuencias pueden alinearse en el tiempo.[2]

A.6 Red Neuronal Convolucional (CNN)

Una red neuronal convolucional es un tipo de red neuronal artificial donde las neuronas corresponden a campos receptivos de una manera muy similar a las neuronas en la corteza visual primaria de un cerebro biológico. Estas redes consisten en múltiples capas de filtros convolucionales de una o más dimensiones. Como redes de clasificación, al principio se encuentra la fase de extracción de características, compuesta de las neuronas convolucionales. Al final de la red se encuentran neuronas sencillas para realizar la clasificación final sobre las características extraídas.[3]

El término convolucional hace referencia a la operación matemática de convolución. Las CNN utilizan la convolución, en vez de la multiplicación de matrices,

[2]https://en.wikipedia.org/wiki/Dynamic_time_warping
[3]https://es.wikipedia.org/wiki/Redes_neuronales_convolucionales

en al menos una de sus capas. En casi todas las redes se aplica una operación denominada *pooling*, que viene a ser una reorganización de píxeles mediante alguna operación de agrupación realizada sobre una ventana en los resultados intermedios que se obtienen tras la convolución. En esencia, la convolución consiste en el desplazamiento de un núcleo, conteniendo una serie de valores, realizando las operaciones de producto y suma [14].

A.7 Perceptrón Multicapa (MLP)

El perceptrón multicapa es una red neuronal artificial formada por múltiples capas, de tal manera que tiene capacidad para resolver problemas que no son linealmente separables [4]. Cada una de las neuronas ocultas o de salida recibe una entrada de las neuronas de la capa previa (conexiones hacia atrás), pero no existen conexiones laterales entre las neuronas dentro de cada capa [12].

[4] https://es.wikipedia.org/wiki/Perceptrón_multicapa

Bibliografía

[1] Carlos-D. Martínez-Hinarejos, Zuzanna Parcheta. Spanish Sign Language Recognition with Different Topology Hidden Markov Models. *INTERSPEECH 2017*, August 20–24, pp. 3349-3353, 2017, Stockholm, Sweden.

[2] Carlos-D. Martínez-Hinarejos, Zuzanna Parcheta. Sign Language Gesture Recognition Using HMM. *Springer International Publishing AG 2017*, L.A. Alexandre et al. (Eds.): IbPRIA 2017, LNCS 10255, pp. 419–426, 2017.

[3] Carlos-D. Martínez-Hinarejos, Zuzanna Parcheta. Sign Language Gesture Classification Using Neural Networks. *IberSPEECH 2018*, 21-23 November, pp. 127-131. 2018, Barcelona, Spain.

[4] Lionel Pigou, Sander Dieleman, Pieter-Jan Kindermans, and Benjamin Schrauwen. Sign Language Recognition Using Convolutional Neural Networks. *Springer International Publishing Switzerland 2015*, L. Agapito et al. (Eds.): ECCV 2014 Workshops, Part I, LNCS 8925, pp. 572–578, 2015.

[5] Juan David Guerrero-Balaguera, Wilson Javier Pérez-Holguín. FPGA-based translation system from colombian sign language to text. *DYNA*, vol 82 no. 189, pp. 172-181. February, 2015.

[6] Michael H. Kuhn, Horst Tomaschewski, Hermann Ney. FAST NONLINEAR TIME ALIGNMENT FOR ISOLATED WORD RECOGNITION. *Philips GmbH Forschungslaboratorium Hamburg*, D-2000 Hamburg 54, F.R.G.

[7] Pavel Senin. Dynamic Time Warping Algorithm Review. *Technical report, University of Hawaii*. December 2008.

[8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[9] Maciej Łuczak Combining raw and normalized data in multivariate time series classification with dynamic time warping. *Journal of Intelligent & Fuzzy Systems*, vol 34, no 1, 373–380, 2018.

[10] Carrillo Aguilar, Roberto. DISEÑO Y MANIPULACIÓN DE MODELOS OCULTOS DE MARKOV, UTILIZANDO HERRAMIENTAS HTK: UNA TUTORÍA. *Ingeniare. Revista chilena de ingeniería*, vol 15, no 1, 18-26, 2007.

[11] Luciano Maldonado. Los modelos ocultos de Markov, MOM. *TELOS. Revista de Estudios Interdisciplinarios en Ciencias Sociales UNIVERSIDAD Rafael Belloso Chacín*, vol 14, no 3, 433-438, 2012.

[12] María V. López, María G. Longoni, Eduardo A. Porcel. MODELOS ESTADÍSTICOS Y CONEXIONISTAS PARA PREDECIR EL RENDIMIENTO ACADÉMICO DE ALUMNOS UNIVERSITARIOS. *Investigación Operativa*, vol 20, no 33, 135-157, 2012.

[13] Rodríguez Rodríguez, J. E., Rojas Blanco, E. A., & Franco Camacho, R. O. CLASIFICACIÓN DE DATOS USANDO EL MÉTODO K-NN. *Revista vínculos*, vol 4 no. 1, 4-18, 2007.

[14] Molinero Lacave, Jaime y Ionut Vaduva, Andrei. Reconocimiento de objetos en imágenes mediante técnicas de aprendizaje profundo: Redes Neuronales Convolucionales. *Trabajo Fin de Grado*, Universidad Complutense de Madrid, 2019.

[15] Douglas Reynolds. Gaussian Mixture Models. MIT Lincoln Laboratory, 1991.

[16] Weibo Liu, Zidong Wang, Xiaohui Liu, Nianyin Zeng, Yurong Liu, Fuad E. Alsaadi. A survey of deep neural network architectures and their applications. *Neurocomputing*, vol 234, 11-26, 2017.