UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Detección y evaluación de ejercicios de rehabilitación para pacientes con la enfermedad de Parkinson

Presentado por Lucía Núñez Calvo
en Universidad de Burgos — 6 de julio de 2022
Tutor: José Francisco Díez Pastor, José Luis Garrido Labrador y José Miguel Ramírez Sanz

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

Agradecimientos

Transmitir mi más sincero agradecimiento a la Universidad de Burgos por haberme permitido utilizar la máquina *gamma* del grupo *ADMIRABLE* para poder desempeñar este proyecto, a mis tutores D. José Francisco Díez Pastor, D. José Luis Garrido Labrador y D. José Miguel Ramírez Sanz por haberme prestado toda la ayuda posible para resolver gran parte de mis dudas y encaminarme hacia nuevas líneas de pensamiento.

También agradecer a mis compañeros Carlos Ortúñez Rojo y Guillermo Saldaña Suárez su amabilidad y paciencia al prestarse voluntarios para la realización de varios vídeos y autorizar su uso en este proyecto.

Finalmente, agradecer a mi gran compañero Jorge Gómez Ortíz por facilitarme su ordenador personal para poder realizar sobre él la aplicación de escritorio y a mi hermana Esther Núñez Calvo, por aguantarse la risa mientras me grababa realizando diferentes ejercicios.

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. José Francisco Díez Pastor, D. José Luis Garrido Labrador y D. José Miguel Ramírez Sanz, profesor del departamento de Ingeniería Informática, área Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. Lucía Núñez Calvo, con DNI 71314978M, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 6 de julio de 2022

Vº. Bº. del Tutor:

Firmado por RAMIREZ SANZ JOSE MIGUEL - 71303106R el día 07/07/2022 con un certificado emitido por AC FNMT Usuarios

D. José Miguel Ramírez Sanz

Vº. Bº. del Tutor:

Firmado digitalmente por GARRIDO LABRADOR JOSE LUIS - 71707244Y
Fecha: 2022.07.06 '23:25:09 +02'00

D. José Luis Garrido Labrador

Vº. Bº. del co-tutor:

Firmado por DIEZ PASTOR JOSE FRANCISCO - 71278385M el día 07/07/2022 con un certificado emitido por AC FNMT Usuarios

D. José Francisco Díez Pastor

Resumen

Actualmente el mundo está caracterizado por un aumento de la esperanza de vida y una población cada vez más envejecida, esto repercute en un incremento de enfermedades, en concreto enfermedades neurodegenerativas, que es de lo que va a tratar este estudio. La más común es el *Alzheimer* y seguido de este trastorno se encuentra la enfermedad de *Parkinson*, afectando a una de cada 100 personas mayores de 60 años [30].

La enfermedad de *Parkinson* es un tipo de trastorno del movimiento que se produce como resultado de la pérdida de células cerebrales productoras de dopamina [20]. Esta enfermedad causa temblores en manos, brazos, piernas, mandíbula y cara, una rigidez en las extremidades y tronco, lentitud de movimientos e inestabilidad postural, o deterioro del equilibrio y la coordinación [36]. Es muy importante que el paciente realice una terapia rehabilitadora que le permita conservar lo máximo posible sus funciones motoras y locomotoras hasta su inevitable pérdida [26]. Y es por esta razón que para poder permitir a los pacientes unas terapias asequibles y de buena calidad se han realizado numerosos estudios que abarcan el problema de desarrollar aplicaciones de ayuda para la enfermedad de *Parkinson* [8].

Este estudio parte de un trabajo previo realizado por los alumnos José Miguel Ramírez Sanz y José Luís Garrido Labrador. En su trabajo fin de máster se obtenía la descomposición de un vídeo en sus diferentes *frames* y de cada uno de ellos se sacaba el esqueleto correspondiente. En este estudio se va a partir de la obtención de cada uno de los esqueletos, que serán almacenados en forma de posiciones, para su posterior análisis.

La nueva funcionalidad añadida permite la identificación de ejercicios. A partir de una serie de vídeos grabados por pacientes y terapeutas, se realizará un análisis entre los movimientos del terapeuta y los de los pacientes. Una vez se haya obtenido el análisis de ambos vídeos, se deberá localizar el vídeo que realiza el terapeuta dentro de la secuencia de ejercicios que realiza el paciente. De esta manera se podrá observar como de correcto o preciso es el ejercicio que realiza el paciente.

Finalmente, para poder visualizar los resultados, se ha implementado una aplicación de escritorio en la que el usuario puede observar un recorte del ejercicio que realiza el paciente.

Descriptores

Visión artificial, Flujo de datos, Big Data, Rehabilitación, Parkinson, Procesamiento de imágenes, Investigación, Docker.

Abstract

Actually the world is characterized by a life hope's increase and an older population, this has a direct impact into a disease's increase, specifically neurodegenerative diseases, this is what this study is going to be about. The most common disease is the *Alzheimer* and besides this one, we have the *Parkinson*, which affects to one of each 100 older than 60 years person [30].

The Parkinson disease if a kind of movement's disorder which produces as a result of the loss of dopamine producing brain cells [20]. This disease produces tremors in the hands, arms, legs, jaw and face, a rigidity in the extremities and the body, slower motions and postural instability, or loss of the balance and the coordination [36]. It's really important that the patient makes a rehabilitation therapy which will let him keep its motor and locomotor functions the best as possible until its unavoidable loss [26]. This is the main reason to allow the patients have an affordable and high quality therapies there have been created many studies which develops applications to help the *Parkinson* disease [8].

This study starts from a previous job made by the students José Miguel Ramírez Sanz y José Luís Garrido Labrador. In their TFM they obtained the decomposition of a video in its differents frames and from each of this ones they get its corresponding skeleton. This study starts from the obtainment of each of the skeletons, which will be stored as possitions, for its further analysis.

The new added functionality allows the identification of exercises. From a videos' series filmed by patients and therapists, there will be made an analysis between the therapist and patient's movements. When the have the analysis' results of both videos, there should be localized the therapist's video inside the exercise's sequence made by the patient. By this way there could be seen how correct or accurate is the patient's exercise.

Finally, to be able to see the results, there has been implemented a desktop application in which the user can watch a clip of the patient's exercise.

Keywords

Computer vision, Data flow, Big Data, Rehabilitation, Parkinson, Image processing, Research, Docker.

Índice general

Índice de figuras

Índice de tablas

Introducción

En la enfermedad de *Parkinson* los síntomas motores representan la base del diagnóstico. Uno de los principales síntomas es la `bradicinesia` [1] pero esta enfermedad también cuenta con otros síntomas no motores como pueden ser trastornos del sueño, sensoriales y neuropsiquiátricos como la depresión [12] [9].

Es una enfermedad que afecta sobretodo a la población de avanzada edad con un promedio de inicio a partir de los 60 años y siendo dos veces más frecuente en hombres que en mujeres. Aunque cabe destacar que aun siendo menos frecuente esta patología en mujeres, una vez que la han desarrollado, experimentan una tasa de supervivencia más baja y una progresión de la enfermedad mucho mayor padeciendo unos síntomas no motores más graves que los de los hombres [28].

Actualmente, no hay cura para la enfermedad de *Parkinson*. Por ello la fisioterapia desempeña un papel crucial en el manejo de los síntomas motores de los pacientes que padecen esta enfermedad. Para poder llevar un seguimiento exhaustivo de la precisión con la que se realizan los ejercicios prescritos, los pacientes han de estar acompañados por un terapeuta, lo que conlleva un desplazamiento de su domicilio varias veces por semana [32]. Contando con que los pacientes suelen ser personas mayores y muchas veces dependientes, para una gran cantidad de familias es complejo adaptarse a una rutina en la que tienen que trasladar o acompañar al paciente hasta su centro de salud más cercano. En muchas ocasiones son familias que residen en el medio rural y necesitan hacer uso del trasporte privado para poder acudir a las visitas del terapeuta. Estas visitas deben hacerse periódicamente y esta es una de las consecuencias de la saturación de los servicios. Desgraciadamente

[1] La bradicinesia es un trastorno caracterizado por la lentitud de movimientos

no existe suficiente personal sanitario que pueda atender a los pacientes con la frecuencia que necesitan, o incluso en ocasiones puede que las sesiones sean con diferentes terapeutas, imposibilitando un buen seguimiento de la progresión de la enfermedad [17].

Por estas razones se pretende crear un sistema por el cual los pacientes puedan realizar los ejercicios cómodamente en su hogar y a través de un análisis posterior determinar si se han realizado correctamente.

Este proyecto surge a partir de un par de trabajos fin de máster creados por los compañeros *José Luis Garrido Labrador* y *José Miguel Ramírez Sanz*. En sus proyectos se ofrece un programa por el cual se obtiene una descomposición en imágenes a partir de un vídeo de referencia. Estas imágenes pueden ser `anonimizadas`[2] para ocultar el rostro de la persona y sobretodo, una vez extraídas las imágenes se realiza una extracción del esqueleto del paciente. Este esqueleto puede ser almacenado como una secuencia, perteneciendo cada valor a una posición del esqueleto respecto de la imagen.

El proyecto que se ha logrado es la búsqueda y comparación de estas secuencias que componen un esqueleto. A partir de todas, o una gran parte de las secuencias que el programa puede sacar de un vídeo completo, se dispondrán a ser almacenadas. Una vez que se tienen almacenadas tanto las secuencias de un ejercicio completo por una parte, como las secuencias de varios ejercicios concretos por otra, se ha implementado un algoritmo que localiza la secuencia que representa el ejercicio específico dentro de la la secuencia compuesta por varios ejercicios, con el objetivo de poder comparar entre entre el ejercicios de referencia que realiza el terapeuta con el mismo ejercicio pero en este caso realizado por el paciente. Hay que tener en cuenta que los pacientes pueden realizar vídeos a una velocidad inferior que el terapeuta, o que los movimientos realizados pueden ser parecidos pero nunca iguales. Todo ello se ha tenido en cuenta para las comparaciones de secuencias.

Finalmente una vez se haya detectado en que punto exacto comienza y termina un ejercicio, por medio de la aplicación de escritorio que ha sido creada para mostrar el funcionamiento del programa, se ha recortado el vídeo original y se ha implementado un código que muestra el ejercicio buscado al usuario.

[2]Expresar un dato relativo a entidades o personas, eliminando la referencia a su identidad.

Una mejora muy interesante sería informar de como de bien o mal realizado está ese ejercicio pero más adelante se detallarán los inconvenientes que han surgido al intentar resolver este problema.

Objetivos del proyecto

2.1. Objetivos generales

- Investigar como implantar el algoritmo *dynamic time warping* en la búsqueda de secuencias.
- Realizar una exhaustiva investigación sobre los algoritmos de comparación de secuencias temporales, para comprobar el inicio y el final de éstas en otras secuencias.
- Investigar múltiples librerías en *Python* que permitan la localización de secuencias multidimensionales lo más parecidas posible a un patrón de referencia, dentro de una secuencia de mayor tamaño.
- Investigar y analizar como se podría clasificar un ejercicio según las partes del cuerpo que estén en movimiento.
- Recopilación de vídeos adecuados para la investigación, tanto para ser empleados en este proyecto como en proyectos futuros, comprobando que son idóneos para su análisis.
- Realizar una aplicación de escritorio por la cual se pueda observar un recorte de un ejercicio concreto de un vídeo con múltiples ejercicios.

2.2. Objetivos técnicos

- Aplicar las técnicas investigadas a las secuencias de esqueletos.

- Desarrollar un algoritmo en *Python* que permita la búsqueda y obtención de secuencias dentro de una de mayor tamaño.
- Desarrollar una web para mostrar el contenido de mi proyecto
- Aprender y poner en práctica el manejo de contenedores *Docker*
- Utilizar un sistema de control de versiones, utilizando para ello la plataforma *Github*.
- Utilizar el sistema de composición de textos LaTeX para la realización de la memoria, anexos y conseguir una cierta destreza.
- Utilizar el editor *Overleaf* para la realización de la memoria y anexos.

2.3. Objetivos personales

En este apartado se van a exponer los objetivos personales que me he propuesto de cara a la realización del proyecto.

- Intentar aportar algún enfoque o técnica novedosa para la mejora de la calidad de vida de pacientes con la enfermedad de *Parkinson*.
- Poner en práctica varios de los conocimientos adquiridos en la carrera.
- Investigar e indagar sobre las tecnologías *Data Mining*.

Conceptos teóricos

El contenido de este proyecto, mayoritariamente está abarcado dentro de los conceptos de *Minería de Datos* y de *Visión Artificial*. A su vez, se abordarán temas relacionados con el análisis de movimientos y cómo adaptar los resultados obtenidos a lo que se desea interpretar. Para poder poner en práctica todo esto se ha necesitado interiorizar una serie de conceptos clave. Muchos de ellos van a ser expuestos a continuación.

3.1. Minería de datos

A día de hoy cada vez es más frecuente la generación masiva de datos de distintas procedencias y que debido a los avances tecnológicos sean almacenados en grandes bases de datos, haciendo que nos encontremos repletos de los mismos. Esta gran cantidad de datos oculta una fuente de conocimiento inmensa.

La minería de datos o *Data Mining* es la técnica que estudia métodos y algoritmos por los cuales trata de recopilar nuevas relaciones y patrones entre grandes volúmenes de datos [29]. Esto tiene como finalidad poder descubrir y comprender mejor los datos para predecir comportamientos futuros, por ello han surgido nuevas técnicas de aprendizaje automático como puede ser la *Inteligencia Artificial*.

3.2. Descubrimiento de Conocimiento en Bases de Datos

El *Descubrimiento de Conocimiento en Bases de Datos*, en inglés *Knowledge Discovery in Databases* con el acrónimo *KDD*, es una disciplina que tiene como objetivo extraer conocimiento de grandes volúmenes de datos, y que sea usado más adelante para resolver distintos aspectos que al usuario se le pueden plantear [13].

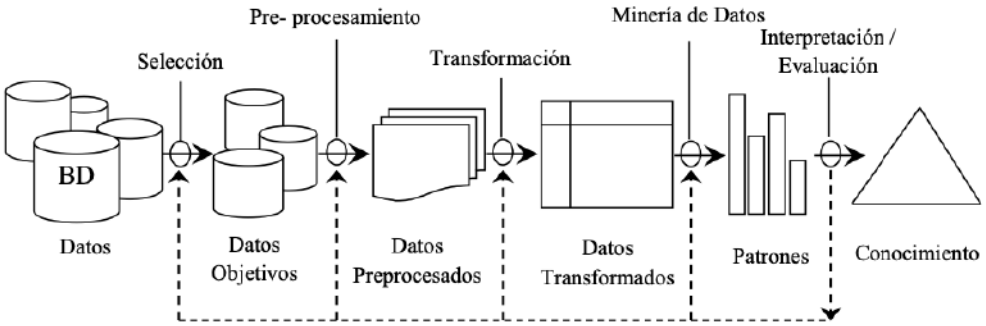


Figura 3.1: Proceso KDD.

Dentro del *KDD* destacan una una serie de pasos imprescindibles, mostrados en la imagen 3.1, para llegar al conocimiento buscado, algunos de ellos son los siguientes:

1. **Selección**, este paso busca la creación de un conjunto de datos sobre el que se va a realizar el descubrimiento. En este proyecto los datos de muestra serán una serie de vídeos.

2. **Pre-Procesamiento**, es la etapa que más tiempo consume. En ella se pretende realizar una trasformación de los datos eliminando las características que el usuario considere innecesarias o trasformando los datos según convenga para su estudio. En este proyecto, de cada vídeo analizado se van a extraer los *frames* que lo componen, de cada *frame* se obtendrá el esqueleto del individuo con la pose del cuerpo. Cada pose se guardará dentro de un fichero con las coordenadas de las posiciones. Este fichero será el conjunto de datos procesado.

3. **Transformación**, En esta etapa se incluye la extracción de características útiles según el objetivo final. En este proyecto esta será la etapa clave, ya que es en este punto del proceso donde se extraerán

las características de cada serie temporal generada y se comparará con otra para poder determinar el objetivo del proyecto.

4. **Interpretación y evaluación**, La evaluación de este proyecto será principalmente visual. El programa informará al usuario sobre los *frames* en los que empieza y termina el vídeo y a través de la aplicación de escritorio se comprobará el resultado.

3.3. Serie temporal

Una *serie temporal* es una secuencia compuesta por datos obtenidos en diferentes instantes de tiempo y ordenados cronológicamente. Estos datos pueden ser extraídos a partir de una única característica, es lo que se conoce como *series temporales univariantes* o a partir de varias características, *series temporales multivariantes* [24].

Una serie temporal compuesta por números naturales tendría la siguiente notación: $X = \{x_1, x_2, ...\}$ o $\{X_k\}$ $k >= 1$ o $X = \{X_t : t \in T\}$

Las series temporales que se analizarán en este proyecto serán obtenidas de los distintos *frames*. Cada uno de los *frames* que componen el vídeo será analizado y en caso de obtener exactamente diecisiete puntos en el esqueleto, se almacenarán sus posiciones. Cada una de estas posiciones representar un valor dentro de la serie temporal final.

Un ejemplo de una buena obtención del esqueleto, y por tanto de las posiciones es la que aparece en la figura 3.4.

3.4. Visión artificial

La *visión artificial* intenta emular la capacidad de reconocer escenas presentes en el mundo real y a partir de ello realizar una interpretación para poder actuar en consecuencia [23]. De la misma forma que nosotros podemos observar a un paciente y clasificar sus movimientos en función de su agilidad, rapidez o destreza, la visión artificial busca que esa misma clasificación pueda hacerla un ordenador.

3.5. OpenCV

OpenCV es una librería libre de visión artificial desarrollada por *Intel*. *OpenCV* significa *Visión Artificial Abierta* o en inglés *Open Computer Vision*. Esta librería es la más popular en el ámbito de la visión artificial y es comúnmente utilizada en campos de investigación sobre detección de objetos, reconocimiento de gestos, detección de movimientos, segmentación y estimación de poses [3].

Figura 3.2: Módulos de la librería *OpenCV*.

Se trata de una librería multiplataforma que está dividida en cinco módulos principales, los cuales se pueden observar en la figura 3.2 y son expuestos a continuación:

1. **Módulo CV**, contiene las funciones principales para el procesamiento de imágenes. Además cuenta con numerosos algoritmos de visión por computador.

2. **Módulo MLL**, es una biblioteca que incluye múltiples clasificadores estadísticos y proporciona herramientas de clasificación y agrupación para lograr el aprendizaje automático.

3. **Módulo HighGUI**, este módulo contiene las funciones básicas para la entrada y salida de vídeos e imágenes.

4. **Módulo CXCore**, este módulo contiene distintas funciones de *OpenCV* y estructuras de datos básicas.

5. **Módulo CvAux**, este módulo contiene tanto áreas extintas como algoritmos experimentales.

3.6. Segmentación

La **segmentación** es el proceso por el cual una imagen es fragmentada en distintas regiones, de esta manera se pueden obtener los diferentes objetos de la escena.

Cabe destacar que en este proyecto es muy importante que el algoritmo elegido para la detección de objetos sepa distinguir a la perfección entre diferentes objetos que puedan aparecer en la escena. De no ser así podría interpretar mal los movimientos del paciente o mezclar las posiciones del esqueleto con las posiciones de otros objetos que reconociese dentro de la imagen.

3.7. Estimación de poses

La **estimación de poses** es la clasificación y obtención de las distintas posiciones que toman las articulaciones y partes del cuerpo humanas en vídeos o imágenes [16]. En esencia, es una forma de detectar un conjunto de *coordenadas* mediante la definición de las articulaciones que componen el cuerpo humano, como pueden ser muñecas, codos, rodillas, etc.

Las dos técnicas más relevantes de detección de poses humanas son las siguientes [34]:

1. **Estimación de poses en 2D**, detecta las ubicaciones de las articulaciones del cuerpo humano, y otra serie de puntos clave, como pueden ser los ojos, la nariz o las orejas, en el espacio 2D. De esta manera las posiciones se almacenan como una lista de coordenadas X e Y sobre cada punto clave.

2. **Estimación de poses en 3D**, en este caso se representan las articulaciones u otros puntos clave del cuerpo humano a través de tres posiciones. La lista de coordenadas contendrá las posiciones X, Y, Z de cada punto clave.

Para la representación de poses humanas se utilizan principalmente tres modelos de estimación que pueden ser observados en la figura 3.3 y son expuestos a continuación [7, 19]:

1. **Modelo cinemático**, este modelo también es conocido como el modelo basado en esqueletos e implementa la obtención de coordenadas a partir

de los puntos clave del cuerpo humano. Es un modelo intuitivo que es ampliamente utilizado para analizar la relación entre las distintas partes que componen el cuerpo humano.

2. **Modelo basado en contornos**, este modelo también conocido como modelo plano y usado para la estimación de poses en 2D, descompone la figura humana en rectángulos y a partir de ellos crea un contorno. De esta forma se obtiene una representación de la forma y apariencia de un individuo.

3. **Modelo basado en volumen**, este modelo se emplea en la estimación de poses en 3D y su representación está basada en aplicar mallas y formas geométricas. A partir de múltiples poses humanas se crea una estimación centrada en el aprendizaje profundo.

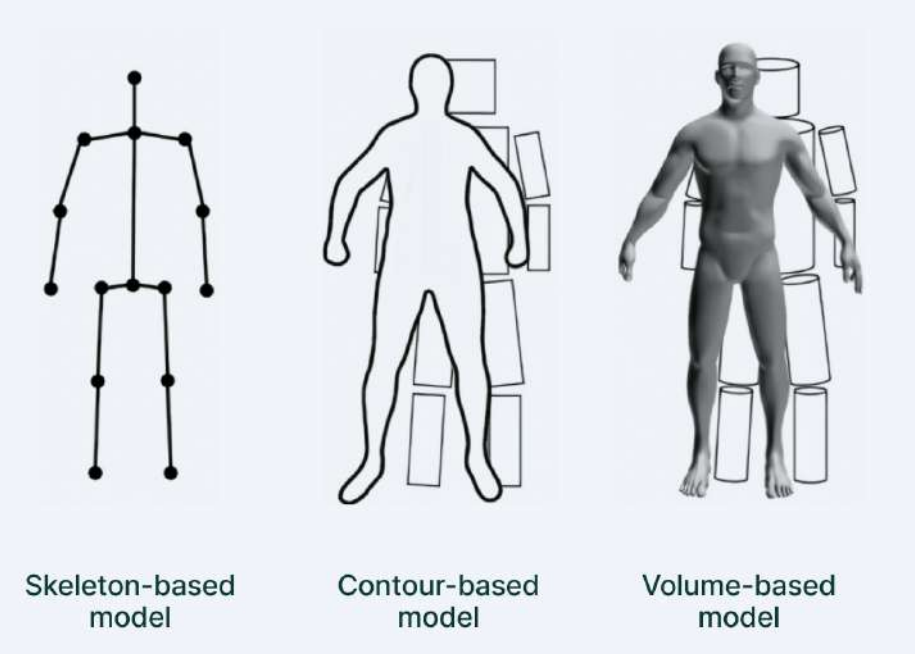
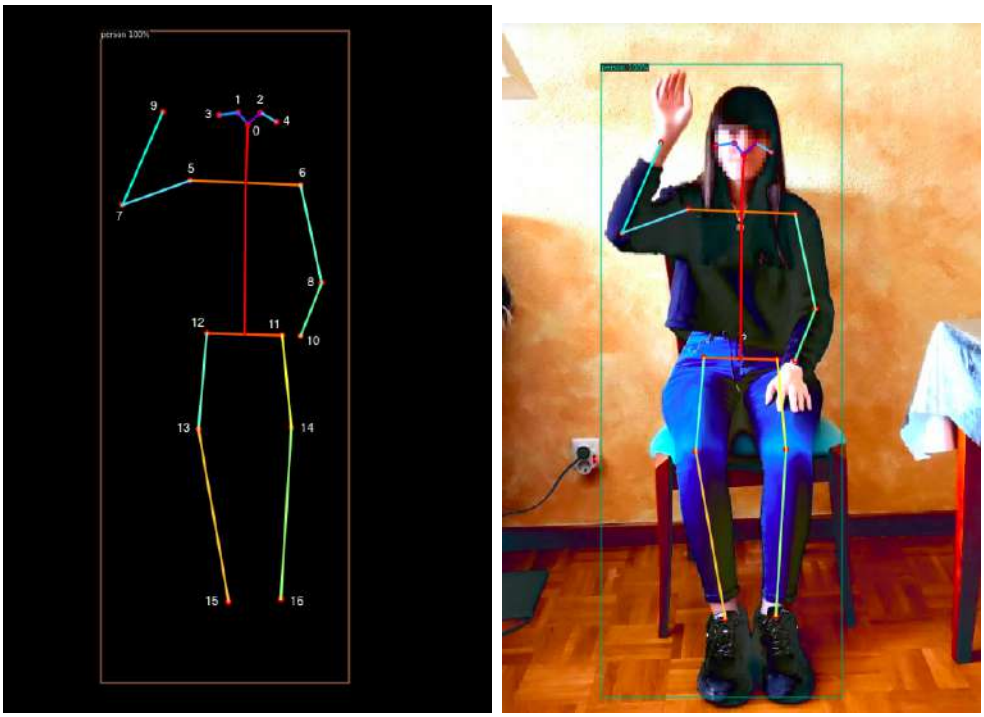


Figura 3.3: Modelos para la detección de poses humanas.

Este proyecto se abarcará utilizando una estimación de poses basada en esqueletos. Por ello es importante conocer como se ha implementado la estructura básica del modelo. En las figuras 3.4a y 3.4b se muestra la creación de esqueletos y la numeración de los puntos clave que se han considerado.



(a) *Esqueleto sobre un fondo negro* (b) *Esqueleto sobre el fondo real*

Figura 3.4: Extracción de esqueletos.

3.8. Redes Neuronales convolucionales

Las **redes neuronales convolucionales** con el acrónimo en inglés *CNN*, son un tipo de Red Neuronal Artificial que proporciona la capacidad de *"ver"* al ordenador. Esto lo realiza procesando cada una de sus capas de forma similar a como lo realiza el *cortex* visual del ojo humano. Por esta razón son ampliamente usadas en tareas de visión artificial o clasificación y segmentación de imágenes [1].

Estas redes están estructuradas alternativamente en capas de convolución y capas de reducción. Las capas de convolución son las encargadas de extraer patrones dentro de los datos de entrada, mientras que las capas de agrupación se encargan de reducir la resolución de los datos de entrada. A su vez, cada una de estas capas está compuesta por un cierto número de mapas de características. En la figura 3.5 se puede observar la estructura de estas redes.

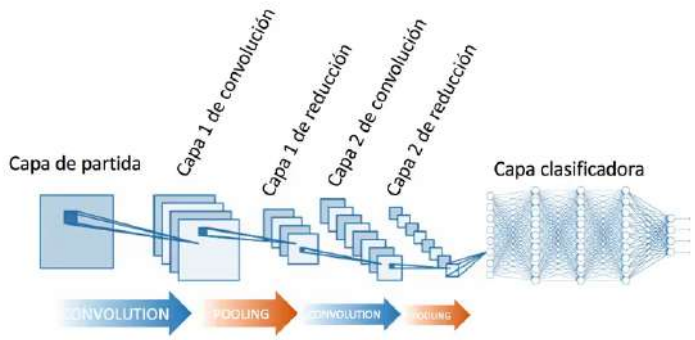


Figura 3.5: Arquitectura de una Red Neuronal Convolucional.

Los principales componentes de la arquitectura *CNN* son [6]:

1. Capa convolucional.
2. Capa de agrupación.
3. Capa de unidades lineales rectificadas *ReLu*.
4. Capa completamente conectada.
5. Capa de pérdida.

Capa convolucional (Convolutional layer)

Es la capa principal en las redes neuronales y pueden usarse desde una a varias capas de este tipo. Las **capas convolucionales** reciben como parámetros un conjunto de filtros que serán aplicados sobre la imagen de entrada generando un mapa de características o mapa de activación bidimensional para cada filtro. Una vez obtenidos los diferentes mapas de características para todos los filtros se genera como salida la capa convolucional.

Mapa de características:

$$h^k_{i,j} = tanh((w^k x)_{i,j} + b_k)$$

Donde w^k es el peso, b_k es el sesgo, x es el valor del píxel específico y $tanh$ especifica la no linealidad de los datos. Los subíndices i, j hacen referencia a las distintas entradas en la matriz que representa el mapa de características.

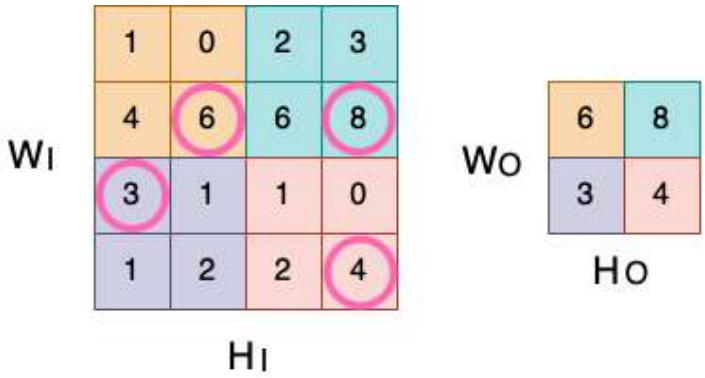


Figura 3.6: Operación de *max-pooling*, filtro 2×2 y S=2.

Capa de agrupación (Pooling layer)

Las capas de agrupación o reducción son comúnmente colocadas entre las distintas capas de convolución y es que su función es la de tomar los mapas de características generados en la capa de convolución previa y agruparlos en una única imagen.

Para poder reducir el número de parámetros que componen la red neuronal se utiliza la operación de *pooling*, actualmente la agrupación mediante *max-pooling*,$f = max$, ha demostrado dar los mejores resultados, es por esa razón que es la más usada pero no hay ningún impedimento en usar cualquier otra función.

Para un volumen $I = (W_I; H_I; D)$ donde $I = I_1...I_D$, la operación de agrupación producirá un volumen de salida $O = (W_O; H_O; D)$ donde $O = O_1...O_D$, deslizando un filtro $2k$×$2k$, que representa el muestreo de respuesta de la función f que se aplica para disminuir la cantidad de parámetros. En la figura 3.6 se puede observar el resultado de la operación de agrupación con $f = max$ Y $S = 2$. El parámetro S indicará los saltos que irá realizando el filtro, es decir, el número de píxeles que se deberá de desplazar el filtro antes de realizar nuevamente la operación. Finalmente esta operación generarán un volumen resultante con unas dimensiones $W_O = W/2, H_O = H/2, D$ y reduciendo en un 75 % la cantidad e parámetros de entrada para poder seguir con el entrenamiento.

Capa de unidades lineales rectificadas ReLu (Rectified Linear Units Layer)

La capa *Rectified Linear Units* o *ReLu* tiene como principal funcionalidad aumentar las propiedades de no linealidad de la función de activación, y no necesitan parámetros, si no que utilizan una función fijada. La función más utilizada es $f(x) = max(0, x)$ aunque se pueden usar otras como $f(x) = tanh(x)$ que permite entrenar con una mayor rapidez o $f(x) = |tanh(x)|$.

Capa completamente conectada (Full connected layer)

Esta capa, como su propio nombre indica, es una capa con una arquitectura completamente conectada por lo que no utiliza propiedades de conectividad local. El único parámetro que se puede configurar de este tipo de capas, es el número de neuronas que componen la capa, y es que cada neuronas de esta capa se encuentran conectadas a todos los mapas de características de la capa anterior.

Como parámetro de salida genera un único vector $1 \times 1 \times K$ que contiene las activaciones calculadas. Al generar una sola dimensión como salida, esta capa indica que tras ella no podrá haber más capas convolucionales. El parámetro K que se obtiene en la salida se usará en la siguiente capa con una función probabilística para realizar la clasificación.

Capa de pérdida (Loss layer)

Esta capa es la que se encarga de la comparación entre las predicciones obtenidas y los valores reales de la imagen. Para esta operación se pueden usar una función de pérdida *softmax* o una función de pérdida euclídea.

Función de pérdida *softmax*

$$\sigma(Z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{3.1}$$

$$z_j = \sum_{k=0}^{d} W_{ik} x_k \tag{3.2}$$

donde:

1. z_j, es el vector de probabilidad.
2. j, es la *iésima* neurona de la capa de pérdida.

3. K, es el número total de neuronas de la capa de pérdida.

4. W_{ki}, son los pesos.

5. x_i, son los valores de entrada de la capa de pérdida.

6. $(z)_j$, es la activación de las K neuronas de la capa de pérdida.

Función de pérdida *euclídea*

$$E = \frac{1}{2K} \sum_{i=1}^{K} ||\hat{y}_i - y_i||_2^2 \tag{3.3}$$

donde:

1. K, es el número total de neuronas de la capa de pérdida.

2. $\hat{y}_i$, son las predicciones de las imágenes.

3. y_i, son los valores reales de las imágenes.

3.9. Dynamic Time Warping

Introducción

Dynamic Time Warping (DTW) es un algoritmo creado para la alineación de secuencias temporales no lineales tratando de encontrar una ruta de alineación óptima [27]. Es ampliamente utilizado en el campo del reconocimiento por voz, la detección y clasificación de movimientos y gestos humanos y el reconocimiento de formas y firmas. Es utilizado en estos ámbitos porque lo que DTW consigue es una alineación de puntos de tiempo muy similares en diferentes instantes de tiempo.

En la figura 3.7 se puede observar como el algoritmo DTW realiza una alineación de puntos entre dos secuencias.

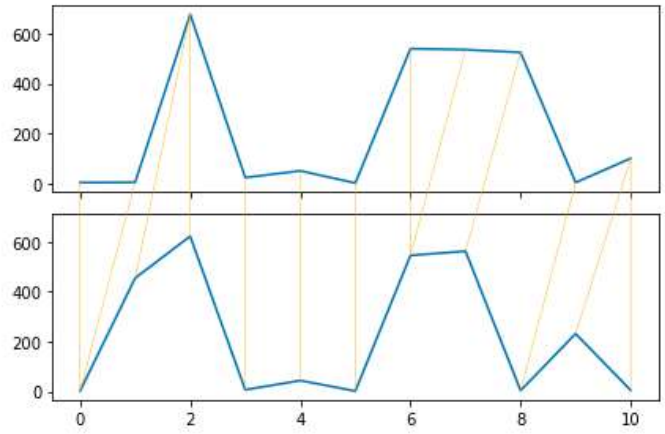


Figura 3.7: Alineación **DTW** entre dos secuencias.

Dadas dos secuencias $X := (x_1, x_2...x_N)$ e $Y := (y_1, y_2...y_M)$ en las que $M \in N$, el objetivo del algoritmo DTWserá encontrar una alineación óptima entre ambas secuencias. En la figura 3.8 se muestra una posible alineación de secuencias señalizada por las flechas moradas bidireccionales.

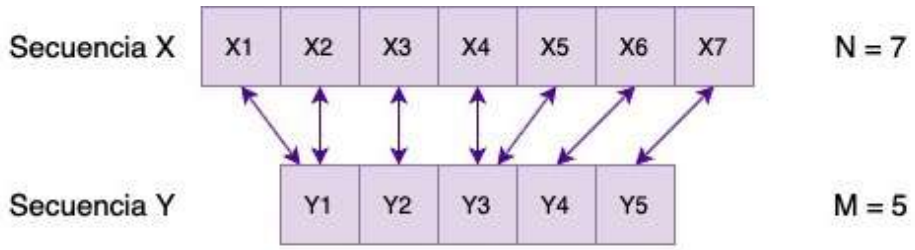


Figura 3.8: Alineación teórica **DTW** entre dos secuencias.

Cada una de las flechas indica una relación entre los valores x_n e y_m para $n \in [1 : N]$ y $m \in [1 : M]$

Camino de deformación DTW

Un camino de deformación determina la alineación entre dos secuencias. Esta alineación es creada a partir de la matriz de costes que crea el algoritmo de DTW. Este camino o ruta de deformación cuenta con una longitud $L \in N$ creando una secuencia $P = (p_1, p_2...p_L)$ que cumpla las siguientes condiciones [5]:

1. Condición de límite, esta restricción especifica que tanto los puntos de inicio como de fin de ambas señales, son emparejados entre ellos.

$$p_1 = (1,1) \quad \& \quad p_L = (N,M) \tag{3.4}$$

2. **Condición de monotonicidad**, esta restricción hace que se cumpla con un orden temporal en la asignación de los elementos.

$$n_1 <= n_2 <= ... <= n_L \quad \& \quad m_1 <= m_2 <= ... <= m_L \tag{3.5}$$

3. **Condición de continuidad**, restringe los saltos en el tiempo y ciñe los cambios a puntos contiguos.

$$l \in [1:L] n_l - n_{l-1} <= 1 \quad \& \quad m_l - m_{l-1} <= 1 \tag{3.6}$$

   o lo que es lo mismo:

$$p_l - p_{l-1} \in \sum := \{(1,0),(0,1),(1,1)\} \tag{3.7}$$

También existen otras restricciones menos frecuentes

1. **Condición de ventana deformada**, esta restricción permite crear una selección de puntos a partir de un valor por defecto, escogiendo solo aquellos que cumplan dicha restricción.

$$l \in [1:L] \quad \& \quad |n_l - m_l| <= (\sigma) \tag{3.8}$$

2. **Condición de la pendiente**, esta restricción permite crear una ruta de deformación a partir de una limitación en cuanto a la pendiente. De esta manera se podrán evitar los movimientos bruscos en un determinado sentido.

Matriz de costes locales

Para poder localizar una ruta de deformación óptima es necesaria la presencia de una matriz de distancias o matriz de costes C representada en la figura 3.9 [39]. Teniendo en cuenta las secuencias $X := (x_1, x_2 ... x_N)$ e $Y := (y_1, y_2 ... y_M)$ si c(x,y) son similares, el valor que proporcionará la matriz será bajo, mientras que si son muy dispares, ese mismo valor c(x,y) será elevado. Evaluando cada una de las combinaciones de la secuencia X

Matriz de costes locales C

	1	3	4	6	5	3
3	2	3	7	8	0	1
4	0	1	5	6	2	1
8	1	0	4	5	3	2
9	3	2	2	3	5	4
1	2	1	3	4	4	3
2	0	1	5	6	2	1

Figura 3.9: Matriz de coses locales.

con la secuencia Y se consigue la **matriz de costes locales** $C \in R_N$xM con $n \in [1:N]$ y $m \in [1:M]$ definida como:

$$C(n,m) := c(x_n, y_m) \tag{3.9}$$

Para calcular esta matriz hace falta especificar una métrica de cálculo la distancia entre dos puntos [21]. En los ejemplos se han realizado las comparaciones con la distancia euclidiana.

Matriz de costes acumulados

Esta nueva matriz $D \in R_N$xM con $n \in [1:N]$ y $m \in [1:M]$ se define a partir de las siguientes condiciones:

1. La primera fila de D está inicializada como:

$$D(1,m) := C(1,m) \tag{3.10}$$

$$m \in [1:M]$$

2. La primera columna de D está inicializada como:

$$D(n,1) := \sum_{k=1}^{n} C(k,1) \tag{3.11}$$

$$n \in [1:N]$$

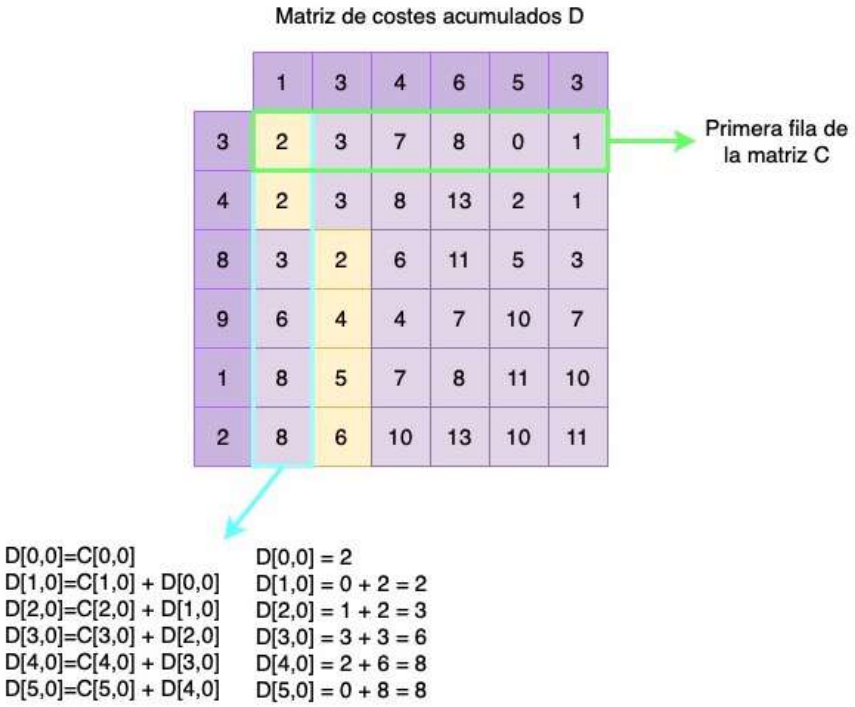


Figura 3.10: Matriz de coses acumulados.

3. El resto de los valores se definen recursivamente a partir de la matriz de costes locales:

$$D(n,m) = C(n,m) + min\left\{(Dn1, m1), D(n1,m), D(n,m1)\right\} \quad (3.12)$$

$$n \in [2:N]$$

$$m \in [2:M]$$

En la figura 3.10 se puede apreciar como a partir de la matriz de costes locales se ha obtenido la matriz de coses acumulados.

Ruta de deformación

Dentro de la matriz de costes acumulados se podrán realizar los siguientes saltos para la elección del camino óptimo:

1. **Movimientos horizontales**, C(n,m) ⇒ C(n,m+1)

2. **Movimientos verticales**, C(n,m) ⇒ C(n+1,m)

3. **Movimientos diagonales**, C(n,m) ⇒ C(n+1,m+1)

Si se tiene una secuencia a localizar, $X := (x_1, x_2...x_N)$ y una secuencia de mayor tamaño $Y := (y_1, y_2...y_M)$ en la que buscar la primera secuencia. Una ruta de deformación será el camino que especifique las alienaciones entre la secuencia a alinear y los valores más similares a ella dentro de la segunda secuencia, que pueden compartir las misma longitud o no. Esta ruta de deformación estará compuesta por tuplas en las que el primer elemento indique la posición de la secuencia de referencia, o la secuencia que se desea alinear y el segundo elemento indicará la posición de la alineación de dicho elemento dentro de la segunda secuencia.

3.10. Búsqueda de secuencias

Dadas dos secuencias $X := (x_1, x_2, x_3...x_N)$ e $Y := (y_1, y_2, y_3...y_M)$ con $N < M$ el objetivo es localizar una secuencia lo más semejante posible a la secuencia X denominada patrón de referencia, dentro de una secuencia de mayor tamaño Y, y que asimismo, informe sobre en que punto se ha localizado dicha secuencia. El objetivo será trabajar con secuencias multidimensionales pero anteriormente se realizarán múltiples ejemplos con secuencias unidimensionales hasta conseguir un óptimo funcionamiento.

En la figura 3.11 se muestra una alineación indicada por las flechas bidireccionales azules, entre el patrón de referencia X de longitud $N = 4$ y una secuencia muy parecida al mismo ubicada en una serie temporal Y de mayor tamaño $M = 10$

Cada una de las flechas bidireccionales azules crea una correspondencia entre dos elementos x_n e y_m para $n \in [1 : N]$ y $m \in [1 : M]$.

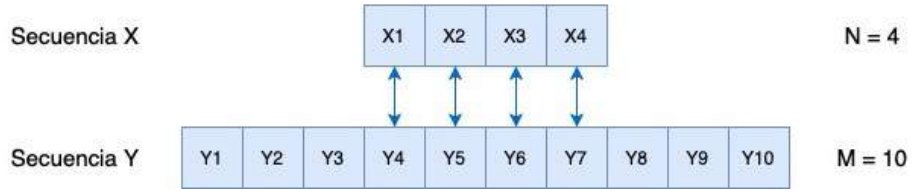


Figura 3.11: Alineación teórica de secuencias.

Formalización del problema

Para modelar matemáticamente nuestro problema de búsqueda de secuencias, consideramos que la longitud de M es mucho mayor que la de N y que existe una matriz de costes locales C dada por $C(n, m) = c(x_n, y_m)$ para $n \in [1 : N]$ y $m \in [1 : M]$ y una matriz de costes acumulados D definida según las restricciones comentadas anteriormente con un tamaño igual al de la matriz de costes locales C. Finalmente para denotar una subsecuencia de Y se utilizarán los índices $a, b \in [1 : M]$ con $a <= b$ y la notación:

$$Y(a : b) := (y_a, y_{a+1}, ...y_b) \tag{3.13}$$

Técnicas y herramientas

En esta sección se explicarán las herramientas que se han puesto en práctica para la realización de este proyecto.

4.1. Python

Este proyecto se ha desarrollado sobre *Python*, un lenguaje de programación de alto nivel, multiplataforma, código abierto y dinámicamente tipado. Es uno de los lenguajes más usados en ámbitos como *Data mining, Data science, Data analytics, Big Data, Inteligencia Artificial y Machine learning*

4.2. Jupyter Notebook

Los cuadernos de *Jupyter* son considerados una aplicación online, diseñados para promover una computación interactiva. Pueden contener código, texto, material multimedia y ecuaciones. Por su gran versatilidad pueden ser utilizados en distintos ámbitos y sus usos más frecuentes son la exposición de resultados y datos, simulación numérica y modelado estadístico.

4.3. Visual Studio Code

Visual Studio Code es un editor de código fuente, proporciona una gran cantidad de opciones para depurar código, permite trabajar con una amplia cantidad de lenguajes de programación como pueden ser *C++, HTML, Java, JavaScript, PHP, R, Python o SQL* entre muchos otros. Además cuenta con la opción de poder tener una plataforma de control de versiones integrada en el editor.

4.4. Git

Git es un *software* de control de versiones de código abierto. Este tipo de sistemas son muy útiles a la hora de organizar proyectos ya que informan al usuario sobre los diferentes cambios realizados.

4.5. Github

Github es una plataforma *online* que permite alojar código de varios proyectos en un mismo repositorio usando el control de versiones *Git*.

4.6. LaTeX

LaTeXes un sistema de composición de textos diseñado principalmente para la creación de documentos con una alta calidad tipográfica [35].

4.7. Overleaf

Overleaf es un editor de LaTeXde código abierto. Integra una amplia variedad de herramientas para poder redactar, editar y publicar documentos en línea. Es una herramienta muy sencilla de utilizar poniendo a disposición del usuario una compilación automática de manera que se puedan ir observando los cambios efectuados a medida que se realiza el documento.

4.8. Librerías de Python

NumPy

NumPy es una biblioteca de *Python* que está especializada en el análisis de datos y en el cálculo numérico, tanto para grandes como para pequeños volúmenes de datos.

Pandas

Pandas [25] es una biblioteca de *Python* escrita como extensión de NumPy utilizada para el análisis y la manipulación de datos. Hay que destacar que es una librería muy cotizada para la manipulación de series temporales.

Tslearn

Tslearn [33] es un paquete de *Python* con varias dependencias, utiliza las bibliotecas *Numpy* y *Scipy* para las manipulaciones de los datos de tipo matriz, las bibliotecas *scikit-learn, Cython y joblib* para un cálculo eficiente y finalmente *keras y tensorflow*.

Este paquete permite la manipulación de conjuntos de datos de series temporales, tanto unidimensionales como multidimensionales, permitiendo el preprocesamiento de estos datos y la extracción de características.

En el ejemplo que se muestra en la figura 4.1 se puede observar como se crea un patrón de referencia denominado *serie2* y a partir de unos valores aleatorios se obtiene una serie temporal de mayor mayor tamaño denominada *serie1* que contiene al patrón de referencia. El objetivo de este ejemplo es mostrar como el algoritmo puede extraer a la perfección el patrón de referencia dentro de la secuencia de mayor tamaño.

```
import random
import numpy as np

array1=np.zeros(500)
array2=np.zeros(500)
serie2=[2,455,623,7,44,2,513,321,67,899,566,6,4,345,44,56,567,321,4,34,456,65,67,788,8,9,432,546,563,4,232,5]

#Se crean unos arrays para añadir ruido antes y después de la secuencia
for i in range(500):
    array1[i]=random.randrange(1000)
    array2[i]=random.randrange(1000)

#Concatenamos
serie1=np.concatenate((array1,serie2,array2), axis=None)


from tslearn.metrics import dtw_subsequence_path

path, dist = dtw_subsequence_path(serie2, serie1)
print("Path: ",path)

path=np.array(path)
a_ast = path[0, 1]
b_ast = path[-1, 1]

print("\n\nLa subsecuencia que debería encontrar es: [2,455,623,7,44,2,513,321,67,899,566,6,4,345,44,56,567,321
print("\nLa subsecuencia encontrada es: ",serie1[a_ast:b_ast+1])
```

```
Path:  [(0, 500), (1, 501), (2, 502), (3, 503), (4, 504), (5, 505), (6, 506), (7, 507), (8, 508), (9, 509), (10, 51
0), (11, 511), (12, 512), (13, 513), (14, 514), (15, 515), (16, 516), (17, 517), (18, 518), (19, 519), (20, 520), (
21, 521), (22, 522), (23, 523), (24, 524), (25, 525), (26, 526), (27, 527), (28, 528), (29, 529), (30, 530), (31, 5
31)]


La subsecuencia que debería encontrar es: [2,455,623,7,44,2,513,321,67,899,566,6,4,345,44,56,567,321,4,34,456,65,67
,788,8,9,432,546,563,4,232,5]

La subsecuencia encontrada es:  [  2. 455. 623.   7.  44.   2. 513. 321.  67. 899. 566.   6.   4. 345.
  44.  56. 567. 321.   4.  34. 456.  65.  67. 788.   8.   9. 432. 546.
 563.   4. 232.   5.]
```

Figura 4.1: Ejemplo del uso de la librería *tslearn*.

dtaidistance

Dtaidistance [15] es una biblioteca de *Python* compatible con *Numpy y Pandas* está destina al uso de datos de series temporales, como es el caso de la Deformación Dinámica de Tiempo (DTW, Dynamic Time Warping).En la figura 4.2 se puede observar un ejemplo que muestra como representar dos series temporales y su alineación *DTW*.

```
:
1  #Representar las series
2  from dtaidistance import dtw
3  from dtaidistance import dtw_visualisation
4
5  serie1=[3,4,678,23,50,1,540,536,525,3,100]
6  serie2=[2,455,623,7,44,2,546,563,4,232,5]
7
8  path = dtw.warping_path(serie1, serie2)
9  dtw_visualisation.plot_warping(serie1, serie2, path)
10 plt.tight_layout()
```

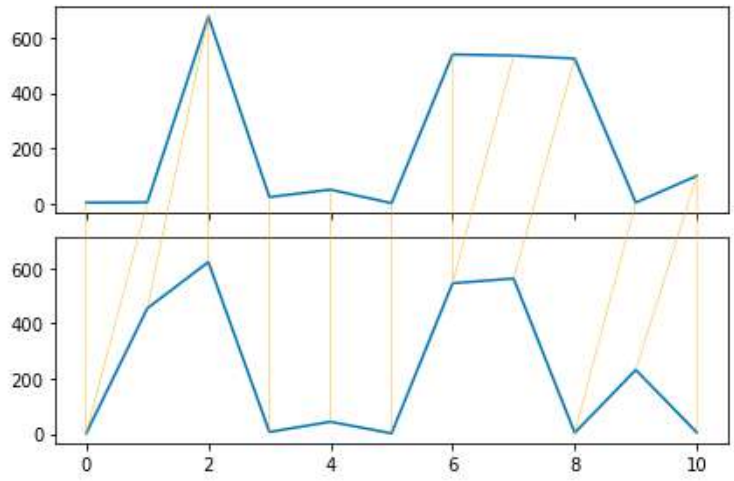


Figura 4.2: Ejemplo del uso de la librería *dtaidistance*.

pydtw

pydtw [14] es una biblioteca de *Python* utilizada para las alineaciones *Dynamic Time Warping*. Esta librería se caracteriza por contener diferentes funciones para el análisis de series unidimensionales y multidimensionales, para el análisis de las primera se usa la función **dtw1d** y para las segundas la función **dtw2d**. En las figuras 4.3 y 4.4 se pueden observar un par de ejemplos de como utilizar estas funciones.

```python
from pydtw import dtw1d

serie1 = np.random.rand(6)
serie2 = np.random.rand(4)
print("Series unidimensionales \nSerie 1: ",serie1,"\nSerie 2: ",serie2)

cost_matrix, cost, alignmend_a, alignmend_b = dtw1d(serie1, serie2)
print("\nMatriz de costes: ",memoryview(cost_matrix).tolist(), "\n\nCoste: ",cost)
print("\nAlineación respecto a la serie 1: ", alignmend_a, "\nAlineación respecto a la serie 2", alignmend_b)
```

```
Series unidimensionales
Serie 1:  [0.34831276 0.2277672  0.37428746 0.75944834 0.14823867 0.31933863]
Serie 2:  [0.3359153  0.28215127 0.48744559 0.14759231]

Matriz de costes:  [[0.012397460253199544, 0.07855894411268494, 0.21769177660964933, 0.4184122244491528], [0.120545
55173340376, 0.0667815281271179, 0.32645991235748606, 0.29786667271574907], [0.15891771173223357, 0.158917711732233
57, 0.17993966087845203, 0.4066348084635858], [0.5824507559053367, 0.6362147795116225, 0.43092046315517274, 0.79179
56926378591], [0.7701273808849589, 0.7163633572786731, 0.7701273808849589, 0.43156682576185446], [inf, 0.7535507107
285184, 0.8844703201852775, 0.6033131431917179]]

Coste:  1.1498179002161812

Alineación respecto a la serie 1:  [0, 0, 1, 2, 3, 4, 5]
Alineación respecto a la serie 2 [0, 0, 1, 1, 2, 3, 3]
```

Figura 4.3: Ejemplo de uso de la función **dtw1d**.

```python
from pydtw import dtw2d

serie1_2D = np.random.rand(5, 3)
serie2_2D = np.random.rand(4, 3)
print("Series multidimensionales \n Serie 1: ",serie1_2D,"\nSerie 2: ",serie2_2D)

cost_matrix, cost, alignmend_a, alignmend_b = dtw2d(serie1_2D, serie2_2D)
print("\nMatriz de costes: ",memoryview(cost_matrix).tolist(), "\n\nCoste: ",cost)
print("\nAlineación respecto a la serie 1: ", alignmend_a, "\nAlineación respecto a la serie 2", alignmend_b)
```

```
Series multidimensionales
 Serie 1:  [[0.55443836 0.27565211 0.71998864]
 [0.64205475 0.79986003 0.05093249]
 [0.98026118 0.85914593 0.49273459]
 [0.03286807 0.6225844  0.37827006]
 [0.71033436 0.7744824  0.31872572]]
Serie 2:  [[0.86921242 0.00355521 0.72924826]
 [0.99417493 0.09634168 0.63527912]
 [0.03393051 0.25932793 0.39095901]
 [0.46331096 0.22179369 0.57686628]]

Matriz de costes:  [[0.4161792539830506, 0.8985651770638896, 1.5145646865647269, 1.692578569277681], [1.48660639525
0288, 1.3961729612159919, 1.780386069082612, 2.316260633057512], [2.381204643596364, 2.172306228164603 5, 2.52119813
20565404, 2.6053306404478533], [3.4793188023616617, 3.2979597813807264, 2.535785806632072, 3.1419676039263873], [4.
367068621574619, 4.098363655416986, 3.3890875978597035, 3.193905799316627]]

Coste:  4.557477800478781

Alineación respecto a la serie 1:  [0, 0, 1, 2, 3, 4]
Alineación respecto a la serie 2 [0, 0, 1, 1, 2, 3]
```

Figura 4.4: Ejemplo de uso de la función **dtw2d**.

simpledtw

simpledtw es una biblioteca de *Python* que cuenta con una implementación dinámica del algoritmo *Dynamic Time Warping*. En la figura 4.5 se muestra un ejemplo se puede observar como a partir de dos series, la función nos devuelve la mejor ruta de alineación entre ambas, el coste de esta alineación, dos listas que contienen los índices de cada una de las series con los que han sido emparejados respecto a la otra serie temporal, y finalmente, una matriz de costes.

```python
import simpledtw

serie1 = np.random.rand(10)
serie2 = np.random.rand(12)

path, cost, mapping_1, mapping_2, matriz_costes = simpledtw.dtw(serie1,serie2)
print("Ruta: ",path, "\nCoste: ",cost)
print("\nCada índice 'i' contiene la lista de índices 'j' en la serie2 con la que se ha emparejado el índice 'i'
print("\nCada índice 'i' contiene la lista de índices 'j' en la serie1 con la que se ha emparejado el índice 'i'
print("\nMatriz de costes: ",matriz_costes)
```

```
Ruta:  [(0, 0), (0, 1), (1, 2), (1, 3), (2, 4), (3, 5), (4, 5), (5, 6), (6, 7), (7, 8), (7, 9), (8, 10), (9, 11)]
Coste:  3.2190279153298738

Cada índice 'i' contiene la lista de índices 'j' en la serie2 con la que se ha emparejado el índice 'i' en la serie
1  [[0, 1], [2, 3], [4], [5], [5], [6], [7], [8, 9], [10], [11]]

Cada índice 'i' contiene la lista de índices 'j' en la serie1 con la que se ha emparejado el índice 'i' en la serie
2  [[0], [0], [1], [1], [2], [3, 4], [5], [6], [7], [7], [8], [9]]

Matriz de costes:  [[0.63417406 0.91827932 1.38430201 2.00665058 2.46989081 2.74351391
  3.25020538 3.48701489 3.99106802 4.43768502 4.88018937 5.35617272]
 [0.92691752 1.25970993 1.0428714  1.32378936 1.44559897 1.51340649
  1.67866734 1.78328844 1.94591096 2.05109735 2.15217108 2.28672381]
 [1.22153224 1.55058211 1.16933475 1.32566063 1.44747025 1.51153521
  1.67866734 1.78141716 1.94591096 2.05296862 2.15404236 2.28859509]
 [1.72429944 1.63704436 1.50395058 1.66027646 1.657494   1.58968649
  1.88681981 1.78406998 2.15406343 2.26112109 2.36406611 2.49861884]
 [2.12089141 2.15873171 1.73239118 1.88871706 1.88315214 1.62572749
  1.85879586 1.78484256 2.05054102 2.25957593 2.46449819 2.60246736]
 [2.38803864 2.77202351 1.83138704 1.98771292 1.97936553 1.71913123
  1.76539212 1.89560945 1.92186886 2.00145902 2.07693653 2.18589304]
 [2.9981112  2.6962454  2.27330823 2.42963411 2.41850425 1.96865283
  2.20172119 1.97810013 2.37556108 2.34438436 2.41986187 2.52881837]
 [3.05833072 3.5543052  2.38124008 2.32170225 2.43241657 2.26898428
  2.03591592 2.31524517 2.04800155 2.1753391  2.3067893  2.40476051]
 [3.50852164 3.52641911 2.66327964 2.76006768 2.60095934 2.35862424
  2.35862424 2.08874229 2.36807154 2.31063541 2.43386031 2.59878951]
 [4.45187997 3.53360065 3.43848659 3.59481247 3.37338383 2.9414316
  3.17449997 2.63473606 2.90197968 3.06643667 3.06232402 3.21902792]]
```

Figura 4.5: Ejemplo del uso de la librería *simpledtw*.

scipy

scipy es una biblioteca libre y de código abierto compuesta por una multitud de herramientas y algoritmos matemáticos. De esta biblioteca se extrae la función **scipy.signal.findpeaks** con la que pueden localizar picos dentro de una señal o secuencia.

Tkinter

Tkinter [4] es un paquete de *Python* que proporciona la interfaz **TK**. Esta herramienta *TK* nos permite desarrollar aplicaciones de escritorio multiplataforma, esto hace referencia a aplicaciones con una interfaz gráfica para distintos sistemas operativos, como Linux, Mac y Windows.

4.9. Detectron

Detectron es una plataforma de detección de objetos creada por *Facebook*. Su objetivo es dar suministro a la sociedad con código de gran calidad y buen rendimiento que se pueda emplear en el ámbito de la investigación de detección de objetos [18].

Algunos de los algoritmos de detección de objetos que *Detectron* incorpora son los siguientes:

1. **Mask R-CNN**, es una ampliación de *R-CNN más rápido*. Este modelo genera cuadros delimitadores y máscaras de segmentación para cada objeto localizado de la imagen [11].

2. **RetinaNet**, este detector esta compuesto por una única capa, aspecto que lo hace más rápido pero algo menos preciso que otros detectores [22].

3. **R-CNN**: Este detector descompone una imagen y hace que estas descomposiciones entren, de forma secuencial, a una red convolucional que se encarga de la clasificación de los distintos objetos añadiéndoles una cierta probabilidad. El inconveniente es que este detector carece de rapidez de procesamiento. Para más información sobre las redes neuronales convolucionales, visitar el apartado 3.8.

4. **R-CNN rápido**, este detector crea dos procesos que pueden ser ejecutados de forma paralela. Por una parte se extraen las características de la imagen y por otra parte se descompone. Finalmente se realiza una proyección de las descomposiciones de la imagen sobre las características extraídas y se hace una clasificación por cada una de ellas.

5. **R-CNN más rápido**, este detector funciona con la misma lógica que el detector anterior con la diferencia de que las descomposiciones de la imagen se realizan una vez extraídas las características de la imagen [31].

6. **R-FCN**, este detector esta basado en regiones y es totalmente convolucional con un cálculo compartido en la plenitud de la imagen.

4.10. Detectron2

Esta biblioteca es la evolución de *Detectron* y *maskrcnn-benchmark* y proporciona algoritmos de detección de objetos de última generación. Una de sus características principales frente al modelo anterior, es que consigue entrenar los datos mucho más rápido [37]. Para mostrar al usuario lo versátil que pueden ser la herramienta *Detectron2* se han creado una serie de imágenes en las que se pueden ver como identifica a diferentes animales, vehículos e individuos.

Como primer comienzo se mostrará una sencilla clasificación de animales usando diferentes algoritmos de detección de objetos. En la figura 4.6 se puede observar como detecta a la perfección los distintos tipos de animales que aparecen en cada fotografía.

(a) *faster_rcnn_R50DC51x* (b) *faster_rcnn_X10FPN3x*(c) *faster_rcnn_R50FPN1x*

Figura 4.6: Detección de animales usando diferentes algoritmos.

Subiendo un poco más el nivel, en la figura 4.7 se pueden observar un par de imágenes que contienen un mayor número de objetos, individuos y animales. En la primera imagen el algoritmo detecta al elefante, a los individuos que salen en ella, e incluso el objeto que porta el animal. Por consiguiente, en la segunda imagen, se puede apreciar como detecta a la perfección a las personas que se encuentran sentadas. Claramente se ven muy borrosas por el desenfoque de la imagen pero aun así, el algoritmo las identifica con un 94 % y 96 % de precisión.

(a) *mask_rcnn_R101DC53x* (b) *faster_rcnn_R50FPN_noaug_1x*

Figura 4.7: Detección de diferentes objetos usando varios algoritmos.

Finalmente en la figura 4.9, se mostrarán unas imágenes más relevantes al objeto de estudio de este proyecto. En ella se puede observar como mediante la implementación de distintos algoritmos de detección de objetos se ha podido extraer el esqueleto de los distintos individuos de las imágenes.

Detectron2 es capaz de localizar los diferentes objetos de la imagen y los algoritmos de detección de poses son capaces de extraer varios esqueletos de una misma imagen. Por ello en la figura 4.8 se pueden observar varios individuos y sus correspondientes esqueletos. Cabe destacar, que si los individuos están demasiado alejados, o si no se les puede detectar completamente, habrá imprecisión a la hora de extraer los esqueletos.

(a) *keypoint_rcnn_R_50_FPN_3x* (b) *keypoint_rcnn_X_101_FPN_3x*

Figura 4.8: Detección de esqueletos.

(a) *faster_rcnn_faster_rcnn_R_101_FPN_3x*

(b) *mask_rcnn_R_50_FPN_1x*

(c) *keypoint_rcnn_R_50_FPN_1x*

Figura 4.9: Detección de esqueletos usando diferentes algoritmos.

4.11. Máquinas Virtuales Dockers

Docker es un proyecto software que proporciona una tecnología de virtualización basada en contenedores y por lo tanto, una virtualización a nivel de sistema operativo. A diferencia de las *Máquinas Virtuales*, los contenedores no albergan un sistema operativo completo, si no que comparten los recursos del sistema sobre el que se ejecutan, reutilizando reutilizan el hardware y el kernel de la máquina sobre la que se hospedan. En la figura 4.10 se puede interpretar la diferencia entre la estructura de una *Máquinas Virtuales* y un *contenedor Docker* [10].

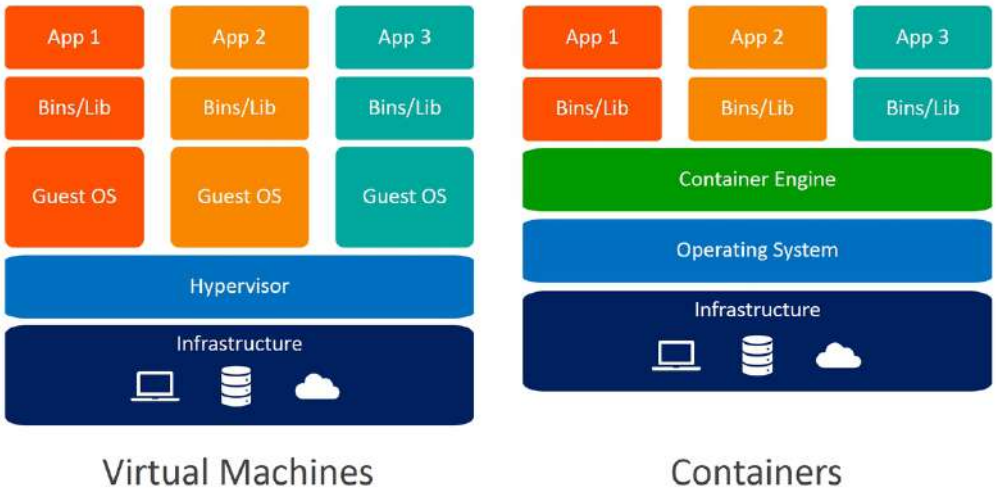


Figura 4.10: Contenedor *Docker* vs Máquinas Virtuales.

Esta tecnología tiene como ventaja la posibilidad de poder desarrollar una aplicación, probarla y ejecutarla desde distintos dispositivos sin tener que perder demasiado tiempo en adaptar el entorno a la aplicación requerida.

Los *contenedores Docker* usan la arquitectura cliente-servidor. El *cliente Docker* se comunica con el demonio de *Docker* a través una *API REST*, una interfaz de red o mediante un *sockets UNIX*. Esta comunicación se realiza para poner en marcha las tareas encomendadas como pueden ser, ejecutar un contenedor o compilar una serie de *imágenes Docker*. Otro cliente con el que cuenta *Docker* es *Docker Compose* que proporciona la posibilidad de poder trabajar con aplicaciones basadas en un conjunto de contenedores.En la figura 4.11 se puede observar una representación de esta arquitectura.

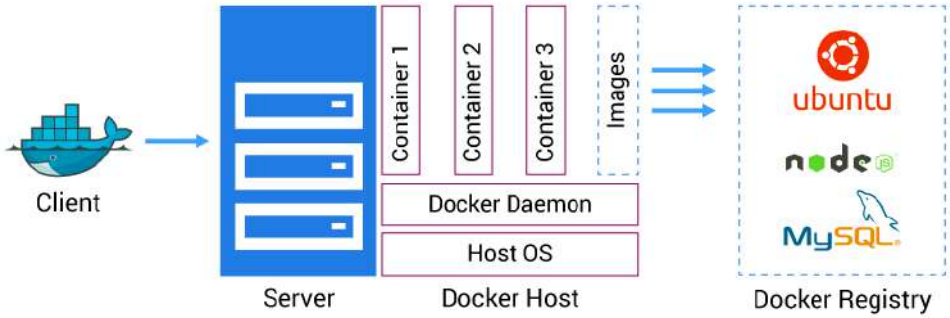


Figura 4.11: Estructura Cliente-Servidor *Docker*.

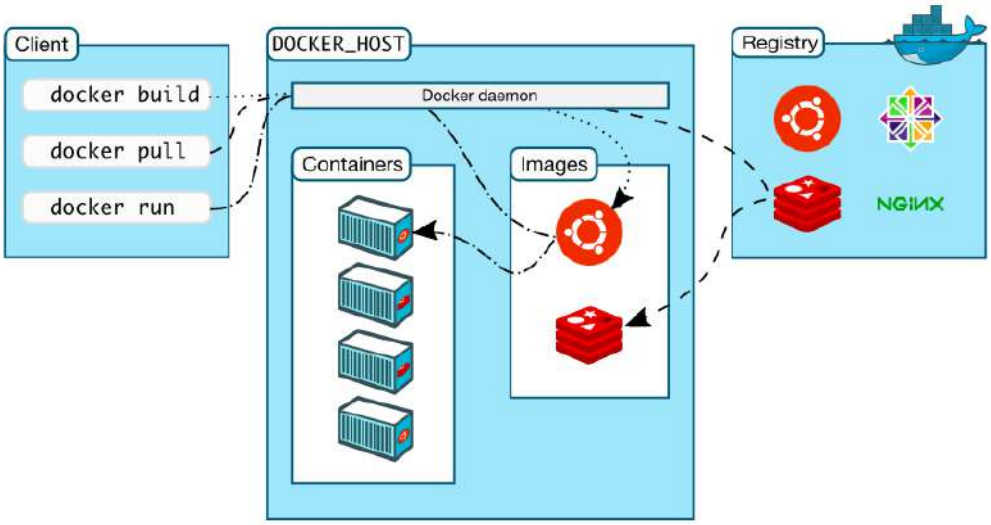


Figura 4.12: Arquitectura *Docker*.

Para entender la arquitectura de *Docker* se dará una breve introducción a los componentes principales, que se muestra en la figura 4.12:

1. **Demonio Docker**, se encarga de atender las peticiones de la *API* y manejar los objetos *Docker* como pueden ser las imágenes o los contenedores. Esto demonios también pueden comunicarse entre sí.

2. **Cliente Docker**, es la manera que tiene el usuario de interactuar con el Docker. Por ejemplo, en la imagen 4.12 se puede observar como el cliente usa los siguientes comandos:

a) *docker build*, es el encargado de permitir al usuario la creación de imágenes Docker.

b) *docker pull*, una vez creada la imagen, este comando permite que sea descargada, por defecto descargará la última imagen creada aunque se puede especificar la imagen que se desea descargar

c) *docker run*, finalmente con este comando se crea un contenedor a partir de la imagen indicada y lo pone en funcionamiento.

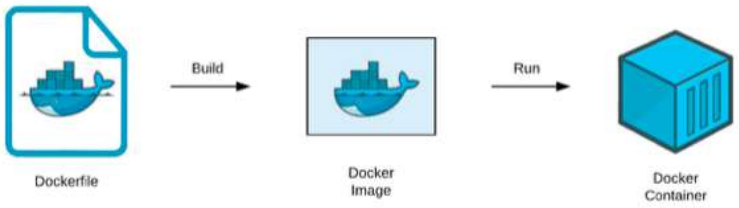


Figura 4.13: Ejecución contenedor *Docker*.

3. **Motor Docker**, permite que sea posible ejecutar los distintos contenedores y las imágenes *Docker*.

4. **Docker Hub**, es un repositorio en el que los usuarios pueden participar compartiendo sus propias imágenes y contenedores *Docker* o adquiriendo los que proporcione la comunidad *Docker*.

5. **Imágenes Docker**, son identidades de software autosuficientes con instrucciones para la creación de contenedores. Para su creación, se empezará creando un *Dockerfile* compuesto por una serie de instrucciones. Cada una de estas instrucciones creará una capa en la imagen. Estas capas hacen que al modificar el *Dockerfile* y reconstruir las imágenes, sólo se reconstruyan las capas que han sido modificadas, haciendo que las imágenes sean tan ligeras pequeñas y rápidas comparadas con otras tecnologías de virtualización.

6. **Contenedor Docker**, un contenedor es una instancia ejecutable de una imagen. Están definidos por estas imágenes y por defecto suelen permanecer aislados de su máquina *host* o de otros contenedores.Se pueden realizar varias acciones sobre ellos utilizando la *API* o la `CLI`[3] como pueden ser:

[3]CLI es una interfaz de línea de comandos (en inglés,command-line interface) que permite a los usuarios escribir comandos a algún programa o al sistema operativo por medio de una línea de texto simple

a) Iniciar un contenedor, *docker container start*.

b) Detener un contenedor, *docker container stop*. Los argumentos que se deben de pasar para que el contenedor se detenga, sería el nombre o el ID del contenedor.

c) Borrar un contenedor, *docker container rm*.

d) Comprobar relaciones entre contenedores, *docker container inspect*. Con este comando se puede consultar información de bajo nivel sobre un contenedor

e) Ejecutar un comando en un contenedor en ejecución, *docker container exec*. Este último comando ha sido muy útil para poder comprobar el correcto funcionamiento del programa.

Aspectos relevantes del desarrollo del proyecto

En este apartado se va a comentar el desarrollo que ha tenido el proyecto. Se expondrán todas las líneas de investigación seguidas, cuál ha sido el problema de cada una y las posibles soluciones.

5.1. Inicio del proyecto

Este proyecto comenzó como un proyecto de investigación sobre la extracción de secuencias mediante el algoritmo **Dynamic Time Warping**. Las secuencias a investigar debían de ser obtenidas mediante un proyecto fin de máster creado por los alumnos José Luís Garrido Labrador y José Miguel Ramírez Sanz.

Por esta razón inicialmente el proyecto no tenía un objetivo concreto más allá de la extracción de las posiciones a partir del proyecto anteriormente mencionado y el análisis de dichas secuencias.

Tras realizar numerosos vídeos y obtener secuencias a partir de distintos puntos del esqueleto humano, se planteó como objetivo principal la localización de ejercicios concretos dentro de secuencias de mayor tamaño y su posterior clasificación según las extremidades del cuerpo que el paciente ejercitase.

5.2. Investigación del proceso a seguir

Una vez establecido el objetivo del proyecto, el proceso a seguir fue el que se puede observar en la figura 5.1: procesar cada uno de los vídeos almacenados, de cada vídeo obtener una descomposición de todos y cada uno de los *frames* que lo componen, de cada *frame* obtener el esqueleto del paciente o terapeuta, almacenar las posiciones del esqueleto, y finalmente analizar y clasificar dichas posiciones. Una vez conocido el proceso a seguir, surgen numerosas cuestiones acerca del mismo: ¿De dónde se van a obtener los vídeos? ¿Qué cantidad de posiciones se precisa almacenar? ¿Qué medios se van a usar para clasificar las secuencias? ¿Qué se considerará un buena clasificación?

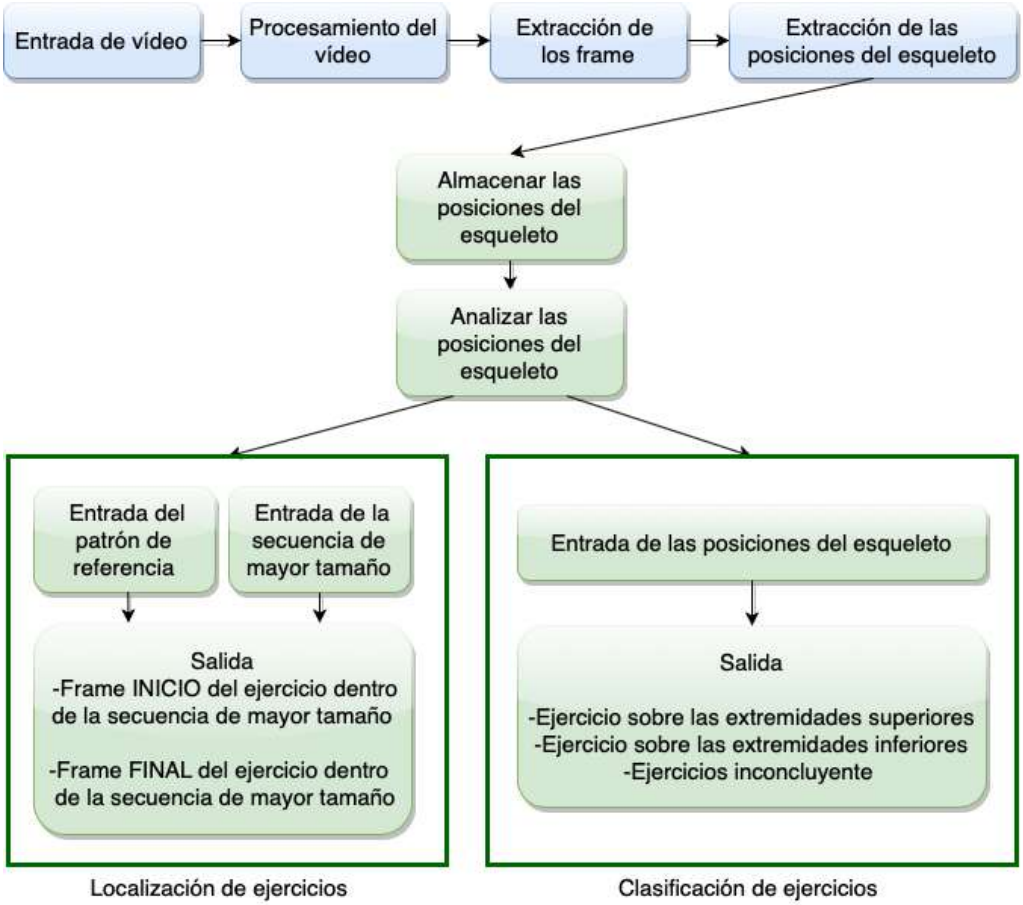


Figura 5.1: Diagrama del flujo del proyecto.

Cabe destacar que como se trata de un proyecto de investigación sobre como sacar información de diferentes vídeos, los resultados obtenidos están influenciados por el tipo de vídeos que se han aportado.

A la hora de responder a las anteriores cuestiones propuestas, se ha hecho una minuciosa investigación sobre múltiples artículos publicados que emplean técnicas similares para el fin planteado. En concreto en el apartado *trabajos relacionados* 5.16, un grupo de investigadores del *Instituto Avanzado de Ciencias de Corea y Technology (KAIST)* realizan un estudio sobre como detectar movimientos de *Tai Chi* utilizando como medida de reconocimiento la distancia **DTW**. Este estudio, que ha sido realizado sobre 21 sujetos, cuenta con las siguientes características:

1. Realiza comparaciones sobre poses en 3D, para ello obtiene las coordenadas X, Y, Z de las diferentes articulaciones y posiciones de los huesos. Para este proyecto, inicialmente se propuso hacerlo con dichas dimensiones pero se descartó rápidamente por la elevada cantidad de tiempo que requerían las ejecuciones.

2. Compara los movimientos redimensionando las series temporales en series unidimensionales, esta característica si que ha sido implantada en este proyecto por los buenos resultados que arroja.

3. Calcula el desplazamiento entre movimientos, es decir, calcula la diferencia que ejerce una extremidad entre el *frame* actual y el *frame* inmediatamente anterior. En este proyecto no se ha calculado la diferencia entre posiciones, pero queda propuesto como líneas futuras realizar un cálculo sobre la diferencia de posiciones y en cuestión del valor obtenido poder encontrar la secuencia de referencia.

Por ello se llegó a la siguiente conclusión, como primera partida sobre la búsqueda de secuencias mediante DTW no es necesario disponer de una amplia cantidad de vídeos, si no, en su defecto, realizar vídeos con una amplia similitud entre los ejercicios que los componen y ver como responde el algoritmo.

Una vez realizados los vídeos con distintos ejercicios en ellos había numerosos factores a tener en cuenta, como: comprobar las características de las posiciones de cada esqueleto, como varían las posiciones entre los diferentes ejercicios, comprobar las diferencias entre las articulaciones que están en movimiento y las que supuestamente deberían de permanecer inmóviles. Este último factor es decisivo en la correcta localización de la

secuencia. Por ejemplo, en un movimiento sobre las extremidades superiores está claro que las posiciones relativas a éstas van a tener una `desviación típica`[4] mucho mayor que las relativas a las extremidades inferiores, pero la cuestión es ¿cómo de diferentes serán esos datos?. Puede que si un paciente realiza ejercicios sobre las extremidades superiores y acaba moviendo las extremidades inferiores, por ejemplo, porque está de pie y se cansa o le cuesta mantener el equilibrio, esta diferencia de posiciones afectará directamente a la localización de la secuencia.

Finalmente, también se realizaron numerosas pruebas sobre que tipo de datos utilizar. Se empezó utilizando un conjunto de datos de tres dimensiones, más adelante se cambió por conjuntos de datos unidimensionales que contenían las posiciones relativas a los ángulos del esqueleto y seguidamente se probó con datos bidimensionales que contenían todas las posiciones extraídas del esqueleto y los ángulos anteriormente comentados. Como se ha especificado, los ángulos son valores unidimensionales por lo que para la correcta comparación de secuencias al ser añadidos a las posiciones bidimensionales del esqueleto, se les tuvo que añadir una segunda dimensión. Esta segunda dimensión fue creada añadiendo un valor de cero en la posición relativa al eje Y. Para la búsqueda de secuencias no es ningún impedimento porque todas las secuencias, tanto las de referencia como la secuencia de mayor tamaño en la que se busca el patrón, tendrían un cero extra en las posiciones Y de los ángulos, por lo que para encontrar similitudes no causaría estragos. Por otra parte si los causaría si lo que se pretende es representar el esqueleto, ya que en este caso localizaría cada uno de los ángulos en una posición del eje X pero siempre en la misma del eje Y, la posición cero.

5.3. Conjunto de Vídeos

Inicialmente el conjunto de vídeos iba a ser aportado por pacientes y terapeutas pero como éstos se demoraban se optó por realizar los vídeos personalmente. Los vídeos con los que se han realizado todas las pruebas son vídeos realizados por tres individuos, uno de ellos la autora del proyecto con sexo femenino y 24 años de edad y los restantes, con un total de dos compañeros, con sexo masculino y 23 años de edad.

Como el objetivo principal de este proyecto es localizar secuencias de movimientos y no evaluarlas, el quien realice los vídeos no es un gran impedimento para lograr el objetivo, lo que si que puede suponer es una falsa esperanza de poder aplicar las técnicas descubiertas a pacientes reales.

[4]Mide la dispersión entre un conjunto de datos numéricos

Todos los individuos con los que se han realizado las pruebas son individuos sanos y jóvenes por lo que los movimientos tienden a ser precisos. Es por esta razón que al no haberse probado con vídeos de pacientes reales no se puede garantizar su funcionamiento en personas con trastornos neurológicos.

Otro aspecto a tener en cuenta es el tipo de movimientos realizados. En los vídeos se han realizado ejercicios cortos y asequibles a personas con la enfermedad de *Parkinson*. Todos los ejercicios muestran movimientos en alguna de las extremidades del cuerpo y son ejercicios cortos para que se puedan realizar sin pausas una vez iniciado el movimiento. De todas formas estos ejercicios han sido propuestos por la autora del proyecto y no son ejercicios establecidos por un terapeuta.

Finalmente comentar que todos los vídeos aportados se han realizado con consentimiento de los voluntarios y han accedido a que puedan ser utilizados para este proyecto.

5.4. Definición del problema

Este apartado consistirá en explicar a grandes rasgos en que ha consistido la búsqueda de secuencias. El primer aspecto a tener en cuenta es la longitud de las mismas. En la figura 5.2 se muestra un ejemplo del problema a plantear. Si tenemos una secuencia de ejercicios provenientes de un paciente, y otra misma secuencia de ejercicios pero en este caso realizados por un terapeuta, es demasiado improbable que ambas secuencias cuenten con la misma longitud. Es por esta razón que se ha elegido la búsqueda mediante **DTW**, que como se ha comentado en el apartado 3.9, busca una alineación óptima entre secuencias de distinto tamaño.

En la figura 5.2 se puede observar como ambas secuencias de esqueletos realizan el mismo movimiento. Según esa figura se puede observar como en la primera secuencia se obtiene un mayor número de esqueletos y por ende una secuencia de una longitud mayor. Esto puede ser provocado por la velocidad del movimiento, si los movimientos son más lentos, capturará un mayor número de *frames* relativos a ese ejercicio. Este ejemplo es simplemente una idea visual para adentrarse en la raíz del problema que se va a plantear a continuación.

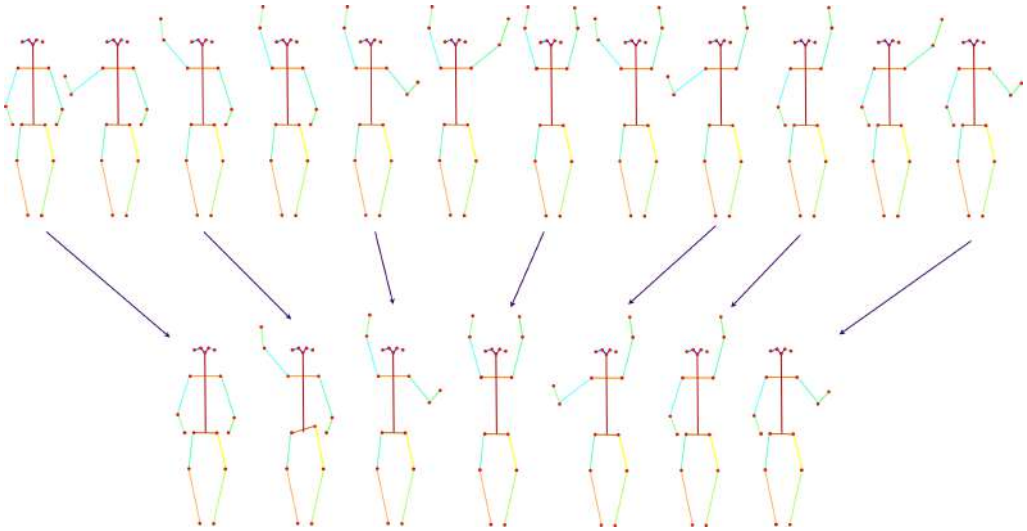


Figura 5.2: Alineación de esqueletos.

Por lo tanto, el problema principal será la búsqueda de la secuencia de ejercicios completa sin que la longitud de la secuencia sea un problema a la hora de realizar la búsqueda. Parece que con *DTW* este problema está solucionado pero en el siguiente apartado se mostrarán algunos de los problemas obtenidos y sus posibles soluciones.

Otro problema a tener en cuenta son los movimientos involuntarios que puede realizar un paciente o individuo.Un ejercicio idóneo de movimientos sobre las extremidades superiores del cuerpo sería el que muestra la imagen 5.3. En esta imagen se puede observar como el paciente mueve únicamente los brazos dejando inmóviles el resto de partes de su cuerpo.

Lógicamente este es un ejemplo simulado que se ha creado para mostrar la dificultad de la búsqueda de secuencias. Por una parte, aunque el paciente se esté lo más concentrado posible en hacer correctamente su ejercicio, es imposible que realice el ejercicio exactamente igual al del terapeuta, pero sobretodo, es muy probable que mueva otras extremidades o partes del cuerpo involuntariamente, o como consecuencia de no mantener un correcto equilibrio. Tanto si se trata de un paciente con la enfermedad de *Parkinson*, o de algún otro individuo, como pueden ser algunos de los lectores de este proyecto, si intentan realizar un ejercicio sobre las extremidades superiores del cuerpo, se acabarán dando cuenta de que en algún momento han movido un pie, la rodilla o han cambiado su posición, sobretodo si se trata de un ejercicio que se realiza de pie.

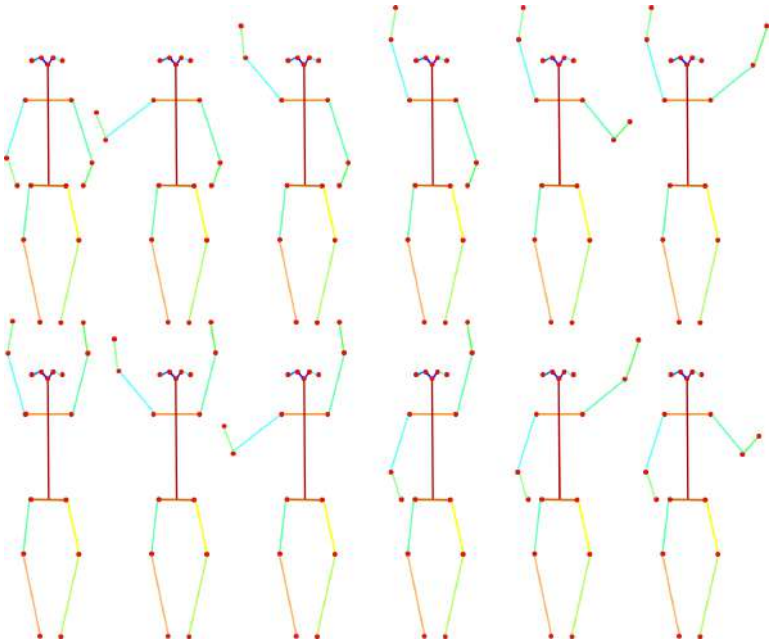


Figura 5.3: Secuencia de movimientos de las extremidades superiores.

El algoritmo que se ha creado para la detección de secuencias tiene en cuenta estas problemáticas. Por otra parte otro problema a tener en cuenta es el tipo de ejercicios. A simple vista para un individuo es muy sencillo diferenciar entre un movimiento de brazos horizontal o vertical pero esto sumado a los inconvenientes anteriormente nombrados hace que para el algoritmo no lo sea tanto.

Imaginémonos que tenemos a un terapeuta que realiza múltiples ejercicios para sus pacientes. Entre los ejercicios que realiza, varios de ellos ejecutan las extremidades superiores del cuerpo. Nuevamente hay que tener en cuenta que este algoritmo está diseñado para pacientes con la enfermedad del *Parkinson* por lo que los ejercicios serán cortos y varios de ellos similares. Uno de los ejercicios será mover horizontalmente los brazos y otro ejercicio será moverlos verticalmente. El paciente realiza ambos ejercicios pero en uno de ellos, por las razones que sea, ejecuta varios movimientos en las extremidades inferiores.

Una vez acabados los vídeos del paciente se procede a analizarlos. El terapeuta quiere comprobar como de bien se ha realizado el movimiento vertical de brazos. Al analizar ambos ejercicios el programa se da cuenta de que tiene por una parte, unos datos de posiciones inferiores muy parecido al del terapeuta pero las posiciones superiores no lo son tanto, y por otra parte

pasa al contrario, tiene uno en que las posiciones superiores son más parecidas a las de referencia pero las inferiores no. ¿Qué secuencia seleccionará el programa como posible solución?

Este ha sido el mayor problema al que se ha enfrentado el algoritmo y es por ello que se ha propuesto en líneas futuras, ver apartado 7.2, realizar la búsqueda de secuencias una vez se tenga clasificado el ejercicio. De esta manera se le pueden otorgar diferentes pesos a las extremidades en movimiento para poder localizarlas de una forma más precisa en la secuencia compuesta por todos los ejercicios.

Esta métrica no se ha implementado en el proyecto ya que la clasificación de ejercicios según las extremidades en movimiento se realizó posterior a la localización de secuencias como un extra, y es simplemente un posible planteamiento de como se podría implementar, por lo que aun no es una métrica del todo fiable.

5.5. Soluciones al problema

En este apartado se detallarán las diferentes lineas de actuación planteadas que se han llevado a cabo para la detección de un algoritmo que funcione con una amplia cantidad de casos, sus inconvenientes y cuales han sido las posibles soluciones o porque han sido descartadas definitivamente. Sobre el algoritmo utilizado se han tenido en cuenta varios aspectos.

El primero de ellos ha sido el tratamiento de los valores nulos. Todos los vídeos que se han procesado, son vídeos idóneos para obtener un buen resultado ya que en ellos aparece un único individuo y en todo momento se pueden apreciar la figura completa del esqueleto. Aún así, en ocasiones *Detectron2* no ha extraído correctamente los esqueletos y ha generado algún valor nulo. Eliminar este tipo de valores es algo demasiado imprescindible para el correcto funcionamiento de nuestro código.

En caso de contar con valores nulos en la secuencia que comprende el conjunto completo de ejercicios, el algoritmo detecta esos valores como el camino más óptimo y devuelve como *frame* de inicio y final del ejercicio de referencia el correspondiente a la posición en la que se han localizado los valores nulos. Es decir, si tenemos una secuencia de longitud N que contiene un valor nulo en la posición k siempre que $k \in N$, localizará k como instante de inicio del ejercicio y ese mismo k como instante final.

Para evitar esta búsqueda errónea de ejercicios se ha procesado a buscar y eliminar todos los nulos en el instante previo al análisis. Estos valores

podrían haber sido sustituidos por una media de los valores próximos pero en una ejecución que generaba 1294 objetos de tipo esqueleto, contando con que cada objeto está compuesto por 27 posiciones y cada posición por sus coordenadas en el eje X y en el eje Y, haciendo un total de 69.876 valores, contando los ceros almacenados en el eje Y de los ángulos, solo se obtuvieron dos valores nulos, por lo que la eliminación de los mismos no parece que vaya a tener mucha repercusión en el correcto funcionamiento del programa.

Otro problema con el que hubo que lidiar fue con la mala obtención del camino óptimo en las pruebas con datos multidimensionales. En las pruebas con datos multidimensionales al contar con datos tan amplios y muchas veces demasiado dispersos entre ellos el algoritmo funcionaba de una manera muy poco precisa.

Finalmente, como se comentaba con anterioridad en el apartado 5.4, el problema principal de este proyecto ha sido la localización de secuencias completas independientemente de su longitud. Esto con la alineación **DTW** parecía estar solucionado pero en ciertas ocasiones no ha sido así. Para explicar el porque de este problema vamos a plantear un ejemplo.

Si contamos con las siguientes secuencias:

1. - Patrón de referencia, [[10,4,45,6],[34,67,2,56],[34,56,78]]

2. - Secuencia en la que buscar, [[564,444,435,2346],[3344,6347,32,5634], [33244,5346,7438],[10,400,465,6], [3454,6547,52,556],[**60,40,80**], [10234,3424,435,634]]

Como se puede observar el patrón de referencia es una matriz compuesta por tres filas y la matriz en la que se pretende buscar dichos valores tiene un total de siete filas, ambas tienen idéntico número de columnas. Lo que el algoritmo de **DTW** realiza para la búsqueda de secuencias es observar el patrón y buscar una similitud dentro de la cadena larga pero, como solo encuentra similitud con una parte del patrón, **[60,40,80]**, es esa la que devuelve. Esto es lo que pasa en la búsqueda de secuencias de algunos de nuestro ejemplos. Como el algoritmo no encuentra coincidencias bastante favorables con el patrón, simplemente las descarta y devuelve una secuencia de una tamaño mucho menor al esperado. En la figura 5.4 se puede observar un ejemplo de lo comentado con anterioridad.

```
print("Longitud del array que contiene el Ejercicio1: ", len(array_angulos1_1))
print("\nLongitud del array que contiene todos los ejercicios: ", len(array_angulos_total))
```

```
Longitud del array que contiene el Ejercicio1:  969

Longitud del array que contiene todos los ejercicios:  2896
```

```
from tslearn.metrics import dtw_subsequence_path

path, dist = dtw_subsequence_path(array_angulos1_1, array_angulos_total)

path=np.array(path)
a_ast = path[0, 1]
b_ast = path[-1, 1]

print("El Ejercicio 1 comienza en el frame =",a_ast)
print("El Ejercicio 1 finaliza en el frame =",b_ast)
```

```
El Ejercicio 1 comienza en el frame = 944
El Ejercicio 1 finaliza en el frame = 1294
```

Figura 5.4: Resultado desfavorable en la localización de secuencias.

Para solucionar este problema se añadirá una tasa de error sobre las posiciones localizadas. De esta manera añadirá unas posiciones extras al comienzo y al final del ejercicio. No es la solución más óptima ya que en algunos casos hará que el ejercicio comience mucho antes de lo esperado o que no llegue a terminar, y en otros casos, cuando encuentre la secuencia prácticamente a la perfección hará que se le añada parte de otros ejercicios.

5.6. Extracción de datos

En este apartado se va a exponer las soluciones planteadas para la extracción de las secuencias. Todo el flujo relativo al procesamiento de vídeos estaba ya implementado en el proyecto que se ha usado como punto de partida pero le faltaba la extracción de las posiciones.

Matizar en este punto que aunque a simple vista la extracción de las posiciones parezca una operación muy sencilla, el familiarizarte con un proyecto nuevo, sobre una tecnología desconocida ha creado en algunas ocasiones múltiples quebraderos de cabeza sobre los errores arrojados.

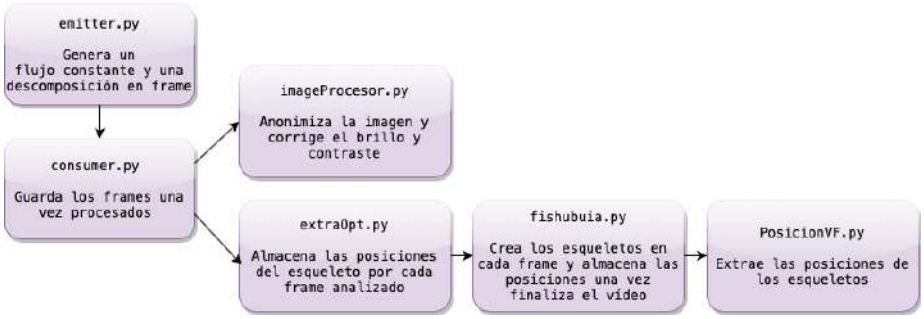


Figura 5.5: Estructura de ficheros.

Finalmente una vez se consiguieron subsanar los problemas por los que no se conseguían almacenar las posiciones correctamente, se procedió a realizar los siguientes cambios sobre los ficheros, en la figura 5.5 se puede observar la estructura de los ficheros y en las figuras 5.6 y 5.7 los cambios realizados sobre los ficheros ya existentes:

1. Fichero **emitter.py**, este fichero es el encargado de lanzar los *frames* y reproducir el vídeo en bucle. Como el proyecto estaba planteado para extraer esqueletos de pacientes en tiempo real, este fichero trata de simular un flujo que nunca termina, es por esta razón que reproduce el vídeo en bucle. Para intentar alterar lo mínimo el funcionamiento del programa, se optó por no parar el vídeo en este punto si no que emitiese un *frame* totalmente negro una vez que el vídeo finalizase.

2. Fichero **PosicionVF.py**, este fichero es el encargado de calcular las posiciones bidimensionales relativas a cada punto clave del esqueleto y de la obtención de los distintos ángulos del cuerpo. En él se han creado nuevas funciones para almacenar los ángulos individualmente, los ángulos junto con el resto de posiciones y las posiciones relativas a las partes superiores e inferiores del cuerpo humano.

3. Fichero **extraOpt.py**, este fichero será el encargado de almacenar las posiciones en cada una de las ejecuciones.

4. Fichero **imageProcesor.py**, finalmente este fichero se encargará de obtener las posiciones que genera el fichero *PosicionVF.py* y enviarlas a *extraOpt.py* para que se encargue de su almacenamiento. Este fichero también proporciona un control de errores y uno de ellos es lanzar un mensaje cuando *Detectron2* no es capaz de extraer el esqueleto, es decir, cuando no encuentra posiciones. Una vez ha entrado en esta excepción, si comprobamos que el *frame* del que no se esta obteniendo el esqueleto es un *frame* totalmente negro, quiere decir que se ha alcanzado el final y del vídeo y por tanto duplicará lo guardado en el fichero *posiciones.pickle* a un nuevo fichero que guardará únicamente las posiciones desde el principio hasta el final del vídeo.

Cabe destacar que hay veces que esta excepción no salta ya que se deshecha el *frame* antes de que llegue a ser analizado. Si esto ocurre y se observa que no aparece el fichero final, lo que se puede hacer es analizar el fichero *posiciones.pickle*. Como se trata de un flujo continuo, cuando el vídeo vuelva a empezar las posiciones serán las mismas por lo que lo único que se tendría que hacer es buscar si la primera posición está repetida en el fichero,

si no es así quiere decir que el programa aún no ha terminado y que necesita seguir procesándose. Sin embargo si encuentra una secuencia tal cual a la primera, está claro que no es una mera coincidencia en los movimientos del paciente si no que el vídeo se está emitiendo de nuevo. A partir de ese momento se puede parar el programa y quedarnos con todas las posiciones hasta la inmediatamente anterior a la secuencia inicial nuevamente generada.

```
if __name__ == "__main__":
    # Creamos un socket (según https://stackoverflow.com/questions/603852/how-do-you-udp-multicast-in-python)
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((IP, PORT))
    connection = s.makefile('wb')

    try:
        video = cv2.VideoCapture(FILE)
        long_video = int(video.get(cv2.CAP_PROP_FRAME_COUNT))   Se obtiene el número de frames del vídeo

        while True:
            # El video de emite siempre en bucle ya que representa
            # Un video real que no pararia
            if countR==(long_video-1): #No generaria el último frame, en su defecto una imagen en negro
                frame = np.zeros([1920,1080,3],dtype=np.uint8)
                frame.fill(0)
            else:                                   Una vez que se alcanza el número de frames
                success, frame = video.read()       se envía una imagen totalmente negra

            if frame is not None:
                if RESIZE != 1.0:
                    frame = cv2.resize(frame, (0,0), fx=RESIZE, fy=RESIZE)

                data = zlib.compress(cv2.imencode('.jpg', frame, [int(cv2.IMWRITE_JPEG_QUALITY), 95])[1])
                sum_ += len(data)
                countR+=1 # Por cada frame leido se aumenta el contador
```

(a) Fichero **emitter.py**

```
class Posicion():

    # paras x: lista con los valores del eje x de los puntos obtenidos con detectron
    #paras y: lista con los valores del eje y de los puntos obtenidos con detectron
    def __init__(self,x,y):
        self.nariz = [x[0],y[0]]
        self.hombroI=[x[5],y[5]]
        self.hombroD=[x[6],y[6]]
        self.cuello = self.calcularPuntoMedio(self.hombroI,self.hombroD)
        self.angCuelloSupI = self.calcularAuxAngulo(self.hombroI,self.cuello,self.nariz)
        self.angCuelloSupD = self.calcularAuxAngulo(self.hombroD,self.cuello,self.nariz)
        self.codoI = [x[7],y[7]]
        self.codoD = [x[8],y[8]]
        self.manoI=[x[9],y[9]]
        self.manoD = [x[10],y[10]]
        self.angCodoI = self.calcularAngulo(self.hombroI,self.codoI,self.manoI,0)
        self.angCodoD = self.calcularAngulo(self.hombroD,self.codoD,self.manoD,1)
        self.angHombroI = self.calcularAngulo(self.cuello,self.hombroI,self.codoI,0)
        self.angHombroD = self.calcularAngulo(self.cuello,self.hombroD,self.codoD,1)
        self.caderaI = [x[11],y[11]]
        self.caderaD = [x[12],y[12]]
        self.cadera = self.calcularPuntoMedio(self.caderaI,self.caderaD)
        self.rodillaI = [x[13],y[13]]
        self.rodillaD = [x[14],y[14]]
        self.angCaderaI = self.calcularAngulo(self.cadera,self.caderaI,self.rodillaI,0)
        self.angCaderaD = self.calcularAngulo(self.cadera,self.caderaD,self.rodillaD,1)
        self.angCaderaTorsoI = self.calcularAuxAngulo(self.cuello,self.cadera,self.caderaI)
        self.angCaderaTorsoD = self.calcularAuxAngulo(self.cuello,self.cadera,self.caderaD)
        self.tobilloI = [x[18],y[15]]
        self.tobilloD = [x[16],y[16]]
        self.angRodillaI = self.calcularAngulo(self.caderaI,self.rodillaI,self.tobilloI,0)
        self.angRodillaD = self.calcularAngulo(self.caderaD,self.rodillaD,self.tobilloD,1)

        self.todosAngulos = self.devuelveAngulos()
        self.todasPos = devuelvePos()

    #Función que permite almacenar los angulos de un esqueleto
    #return vector con los posiciones de los angulos del esqueleto
    def devuelveAngulos(self):
        return [self.angCuelloSupI, self.angCuelloSupD, self.angCodoI, self.angCodoD, self.angHombroI, self.angHombroD, self.angCaderaI, self.angCaderaD, self.angCaderaTorsoI, self.angCaderaTorsoD,
                self.angRodillaI, self.angRodillaD]

    #Función que permite almacenar todas las posiciones relativas a un esqueleto
    #return matriz con las posiciones del esqueleto.
    def devuelvePos(self):
        return [self.nariz, self.hombroI, self.hombroD, self.cuello, [self.angCuelloSupI,0], [self.angCuelloSupD,0], self.codoI, self.codoD, self.manoI, self.manoD, [self.angCodoI,0],[self.angCodoD,0],
                [self.angHombroI,0],[self.angHombroD,0],self.caderaI,self.caderaD,self.cadera,self.rodillaI ,self.rodillaD ,[self.angCaderaI,0],[self.angCaderaD ,0],[self.angCaderaTorsoI ,0],
                [self.angCaderaTorsoD ,0],self.tobilloI ,self.tobilloD ,[self.angRodillaI,0],[self.angRodillaD,0]]

    #Función que permite almacenar parte de las posiciones superiores del esqueleto
    #return matriz con las posiciones superiores del esqueleto
    def devuelvePosSuperiores(self):
        return [self.nariz,self.hombroI,self.hombroD,self.cuello,[self.angCuelloSupI,0],[self.angCuelloSupD,0],self.codoI,self.codoD,self.manoI,self.manoD,[self.angCodoI,0],[self.angCodoD,0],
                [self.angHombroI,0], [self.angHombroD,0]]

    #Función que permite almacenar parte de las posiciones inferiores del esqueleto
    #return matriz con las posiciones inferiores del esqueleto
    def devuelvePosInferiores(self):
        return [self.caderaI, self.caderaD,self.rodillaI, self.rodillaD,[self.angCaderaI,0], [self.angCaderaD,0],self.tobilloI, self.tobilloD ,[self.angRodillaI,0] ,[self.angRodillaD,0]]
```

(b) Fichero **PosicionesVF.py**

Figura 5.6: Ficheros *emitter.py* y *PosicionesVF.py* modificados.

```
ia = Interfaz()

def opt(key, value, output="/"):
    pos, esq, ang, posT  = ia.obtenerPosicion(value, 2)
    with open('posiciones.pickle', 'ab') as f:
        pickle.dump(posT, f)
    return key, esq
```

(a) Fichero **extraOpt.py**

```
#Obtener la posición y la imagen procesada de una imagen
#param imagen: imagen a procesar
#param codigo: 0=sin imagen; 1=imagen normal; 2=esqueleto en blanco; 3 o superior = esqueleto en negro
def obtenerPosicion(self,imagen,codigo=2):
    #Se obtiene la predicción
    output = self.predictor(imagen)
    pkP = output.get("instances").pred_keypoints

    #Si no se encuentran los puntos lanzar excepción
    if len(pkP)==0:
        #Comprobamos antes si todos son 0, es decir, si se trata de la imagen en negro
        if (np.count_nonzero(pkP) == 0)
            shutil.copy('/mnt/data/posiciones.pickle'","/mnt/data/posicionesFINALES.pickle'")
        else:
            raise Exception("No se ha detectado posicion")
    if len(pkP[0])==17:
        x = pkP[0][:,0].cpu().numpy()
        y = pkP[0][:,1].cpu().numpy()
        pos = Posicion(x,y)
        ang = pos.todosAngulos
        posT = pos.todasPos

    #Si no existen exactamente 17 puntos la posición es errónea
    else:
        raise Exception("Posición errónea")

    #Dependiendo del codigo se devuelve o no la imagen
    if codigo==0:
        self.__borrarMemoria()
        im=None
    else:
        if codigo==1:
            visualizer = Visualizer(imagen[:,:,::-1], MetadataCatalog.get(self.cfg.DATASETS.TRAIN[0]), scale=1.2)
        elif codigo==2:
            visualizer = Visualizer(np.full(imagen.shape,255)[:,:,::-1], MetadataCatalog.get(self.cfg.DATASETS.TRAIN[0]), scale=1.2)
        else:
            visualizer = Visualizer(np.zeros(imagen.shape)[:,:,::-1], MetadataCatalog.get(self.cfg.DATASETS.TRAIN[0]), scale=1.2)

        vis = visualizer.draw_instance_predictions(output["instances"].to("cpu"))
        im = vis.get_image()[:, :, ::-1]
        self.__borrarMemoria()
    return pos,im,ang, posT
```

(b) Fichero **fishubuia.py**

Figura 5.7: Ficheros *extraOpt.py* y *fishubuia.py* modificados.

Otro aspecto a tener en cuenta en la extracción de las secuencias, es el tiempo que conllevan. En un sistema ideal hay que ejecutar el programa y esperar a que nos saque el total de posiciones por vídeo. En un vídeo de unos dos minutos esta acción durar alrededor de los 20 minutos, todo dependiendo de factores externos al programa como pueden ser la conexión a la red, etc. Como hemos comentado esto sucedería en un sistema ideal, pero en el apartado *Manual del programador* de los anexos se exponen numeroso aspectos que comprometen el normal funcionamiento del programa. El más tedioso de ellos es la caída de contenedores. Para este proyecto se ha usado el equipo *gamma* del grupo *ADMIRABLE* de la Universidad de Burgos, y se ha comprobado que hay ocasiones en las que al haber más de un usuario realizando tareas sobre el mismo o por otros motivos que en ocasiones se desconoce su causa, provocan que se caiga alguno de los contenedores

en ejecución lo que provoca un efecto dominó sobre el resto, abortando el correcto funcionamiento del programa. Este aspecto es clave para comprender algunas de las dificultades surgidas durante la realización del proyecto. En ocasiones era prácticamente imposible acabar correctamente una ejecución lo que demoraba el análisis de las secuencias al no poder ser obtenidas.

5.7. Metodología de estudio

En este apartado se hará una mera introducción de los distintos planteamientos y fases que se han llevado a cabo a lo largo del proyecto. Para poder contrastar muchos de los aspectos que se van a comentar a lo largo de las secciones es recomendable revisar los *notebooks* ubicados en el directorio **src/pruebas**.

La **primera fase** comprende el estudio de las diferentes librerías aportadas en la sección 4.8. De todas las librerías localizadas sólo algunas fueron útiles para la detección de secuencias. Y es que antes de probarlas sobre datos reales, debían de arrojar buenos resultados sobre datos ficticios, ya que si en secuencias de tamaños relativamente pequeños no se obtenían buenos resultados, en secuencias de mayor tamaño iban a ser desastrosos. Finalmente, para concluir esta fase, se realizaron numerosos ejemplos de como la librería escogida es capaz de encontrar no una secuencia igual, si no una secuencia lo más parecida posible al patrón de referencia. Estos ejemplos se pueden observar en el *notebook* **C3_Búsqueda_y_agrupación_de_secuencias_SIMILARES.ipynb**.

Tras seleccionar la librería a utilizar en el proyecto, la **segunda fase** consistió en analizar la posibilidad de almacenar no una, si no varias rutas óptimas. La razón por la que se planteó esta opción es la siguiente. Como ya se ha comentado anteriormente, la búsqueda de una secuencia completa crea una deficiencia en el longitud de la secuencia esperada. Por ende, un planteamiento que podía resultar interesante es el siguiente. A partir de una secuencia, se sacan las posiciones relativas al inicio y al final de dicha secuencia, con estas posiciones calculamos por separado todas las rutas óptimas del inicio y el final dentro de la secuencia de mayor tamaño, es decir, buscamos todos los posibles inicio y finales. Hasta este momento se tienen almacenadas una serie de posiciones de donde puede empezar el ejercicio concreto dentro de la secuencia larga y donde puede finalizar. Con esa cantidad de posiciones se efectuó un análisis y se obtuvieron los resultados que se comentarán en la sección 5.9.

```
tensor([[[5.1499e+02, 5.5585e+02, 2.6129e+00],
         [5.4228e+02, 5.3137e+02, 2.1253e+00],
         [4.8626e+02, 5.3425e+02, 1.2204e+00],
         [5.8105e+02, 5.5297e+02, 1.0150e+00],
         [4.5610e+02, 5.6017e+02, 7.3380e-01],
         [6.5717e+02, 7.0128e+02, 4.5321e-01],
         [3.9291e+02, 7.2576e+02, 3.0847e-01],
         [8.0510e+02, 7.8048e+02, 5.7422e-01],
         [3.6275e+02, 9.2159e+02, 4.7190e-01],
         [7.1175e+02, 6.2352e+02, 8.5374e-01],
         [4.1302e+02, 1.0512e+03, 8.5506e-01],
         [6.3419e+02, 1.0541e+03, 2.4926e-01],
         [4.6041e+02, 1.0670e+03, 1.8941e-01],
         [6.5430e+02, 1.2528e+03, 7.7593e-01],
         [4.4174e+02, 1.2528e+03, 5.4033e-01],
         [6.1839e+02, 1.6214e+03, 3.5414e-01],
         [5.0637e+02, 1.6243e+03, 4.1325e-01]]])
```

Figura 5.8: Datos obtenidos del fichero *fishubuia.py*.

Una vez elegida la librería a utilizar para el análisis de la secuencia y la primera métrica de prueba que fue la de intentar encontrar la secuencia al completo, se decidió probar con datos reales. Inicialmente se hizo uso del elemento *Posicion* que genera el fichero *fishubuia.py* a partir de las posiciones extraídas mediante el fichero *PosicionVF.py*. Por cada posición localizada genera unos datos como los mostrados en la figura 5.8. Tras varias pruebas sobre distinto tipo de algoritmos fue descartada por varias razones:

1. Al contar con tres dimensiones generaba una cantidad excesiva de datos.

2. Las ejecuciones consumían una cantidad de tiempo inviable. Para detectar un ejercicio de una duración de 32s dentro de un vídeo completo de 1:37s, las ejecuciones sobrepasaban los 50, 60 minutos de análisis.

3. Por si fueran pocas las razones para no utilizar este tipo de datos, los resultados obtenidos estaban lejos de los esperados.

Tras comprobar que este tipo de datos no fueron para nada los esperados comenzó la **tercera fase**. En esta fase se obtuvieron los datos completos a partir de los métodos creados en el fichero *PosicionVF.py*. A partir de dichos datos se procedió a implementar la búsqueda de varios posibles inicios y finales de ejercicios concretos dentro de la secuencia de mayor tamaño. Y además se usó el mismo procedimiento en la **cuarta fase** para localizar la secuencia central del ejercicio.

Finalmente y tras ver que los resultados obtenidos estaban demasiado lejos de los esperados, se puso comienzo a la **quinta fase**. En esta fase se extrajeron dos conjuntos de datos, por una parte, se obtuvieron todos los datos relativos a los ángulos y por otra parte todo el conjunto de posiciones, que abarcaba tanto los ángulos como los puntos clave del esqueleto humano. Para ello se utilizaron las nuevas funciones, anteriormente nombradas, que contiene el fichero *PosicionVF.py*. Con estos conjuntos de datos se realizaron dos análisis por separado que se pueden comprobar en los *notebooks*, **C6_Pruebas_busqueda_angulos.ipynb, C7_Pruebas_busqueda_posiciones.ipynb** y **C8_Pruebas_recortando_frames.ipynb,**, todos buscaban la secuencia de ejercicios concretos dentro de la secuencia de mayor tamaño.

5.8. Primera fase: Análisis de librerías

En el apartado *Técnicas y herramientas* 3.10 se comentan a grandes rasgos muchas de las librerías analizadas para poder ser aplicadas a la búsqueda de secuencias, en concreto a la búsqueda posiciones de esqueletos. En este apartado vamos a comentar las más relevantes.

Para escoger una buena librería se han tenido en cuenta los siguientes factores:

1. Debe de permitir búsquedas de secuencias unidimensionales y multidimensionales.

2. Debe de responder correctamente con grandes volúmenes de datos.

3. El resultado arrojado debe de ser preciso, descartando aquellos resultados que contengan demasiado ruido.

Por supuesto otro aspecto a tener en cuenta es que todas las librerías a utilizar sean capaces de calcular la distancia *DTW*. En esta primera fase, para identificar las secuencias se probó con la librería *dtaidistance*. Esta librería permite trabajar con series temporales unidimensionales y multidimensionales pero fue descartada por la baja precisión con la que se obtenían los resultados. En la figura 5.9, se puede observar claramente la razón por la cual no se llegó a utilizar, y es porque localiza la secuencia con demasiado ruido, no encuentra únicamente el patrón de referencia si no que lo encuentra dentro de cadenas con un tamaño demasiado elevado.

```python
from dtaidistance import dtw
from dtaidistance import dtw_visualisation as dtwvis

d, paths = dtw.warping_paths(serie2_menor, serie1_menor, window=25, psi=2)
best_path = dtw.best_path(paths)

best_path=np.array(best_path)
a_ast = best_path[1, 1]
b_ast = best_path[-1, 1]

print("a* =",a_ast)
print("b* =",b_ast)
print("La subsecuencia que debería encontrar es: [2,455,623,7,44,2,546,563,4,232,5]")
serie1_menor[a_ast:b_ast+1]
```

```
a* = 3
b* = 28
La subsecuencia que debería encontrar es: [2,455,623,7,44,2,546,563,4,232,5]

array([711., 881., 820.,  86., 608., 343., 669.,   2., 455., 623.,   7.,
        44.,   2., 546., 563.,   4., 232.,   5.,  55., 524., 281., 156.,
        21., 752., 564., 172.])
```

Devuelve una subsecuencia en la que se encuentra el patrón de referencia, pero hay demasiados valores antes y después de dicho patrón

Una vez comprobado que el resultado no ha salido como se esperaba, se prueba con un conjunto de datos de mayor tamaño

```python
d, paths = dtw.warping_paths(serie2, serie1, window=25, psi=2)
best_path = dtw.best_path(paths)

best_path=np.array(best_path)
a_ast = best_path[1, 1]
b_ast = best_path[-1, 1]

print("a* =",a_ast)
print("b* =",b_ast)
print("La subsecuencia que debería encontrar es: [2,455,623,7,44,2,546,563,4,232,5]")
serie1[a_ast:b_ast+1]
```

```
a* = 2
b* = 1029
La subsecuencia que debería encontrar es: [2,455,623,7,44,2,546,563,4,232,5]

array([291., 852., 324., ..., 818.,  76., 904.])
```

Saca una secuencia excesivamente larga. Esta técnica no se puede aplicar para conjuntos de datos de gran tamaño

Figura 5.9: Ejemplo de uso de la librería *dtaidistance*.

Seguidamente se probó con la librería **tslearn**. Como se puede observar en la figura 5.10, esta librería es capaz de hallar una secuencia similar al patrón de referencia, devolver la posición en la que ha sido encontrada e informar sobre el grado de similitud que tiene la secuencia encontrada respecto del patrón de referencia.

Para constatar que se podía aplicar esta biblioteca al problema planteado se crearon diversos ejemplos con series de datos unidimensionales y multidimensionales y se observó que en todas las ocasiones arroja los resultados deseados.

Una característica a tener en cuenta que más tarde será utilizada es que, aunque la librería funcione tanto con datos unidimensionales como multidimensionales, al hacerlo con estos segundos, y sobretodo con grandes conjuntos de datos, las ejecuciones consumen una gran cantidad de tiempo.

```
from tslearn.metrics import dtw_subsequence_path

path, dist = dtw_subsequence_path(serie2_menor, serie1_menor)
print(path)
print(dist)

path=np.array(path)
a_ast = path[0, 1]
b_ast = path[-1, 1]

print("a* =",a_ast)
print("b* =",b_ast)
print("La subsecuencia que debería encontrar es: [2,455,623,7,44,2,546,563,4,232,5]"
serie1_menor[a_ast:b_ast+1]
```

```
[(0, 10), (1, 11), (2, 12), (3, 13), (4, 14), (5, 15), (6, 16), (7, 17), (8, 18), (9,
19), (10, 20)]
0.0
a* = 10
b* = 20
La subsecuencia que debería encontrar es: [2,455,623,7,44,2,546,563,4,232,5]

array([  2., 455., 623.,   7.,  44.,   2., 546., 563.,   4., 232.,   5.])
```

Con una colección de datos pequeña, encuentra el patrón a la perfección

```
path, dist = dtw_subsequence_path(serie2, serie1)
print(path)
print(dist)

path=np.array(path)
a_ast = path[0, 1]
b_ast = path[-1, 1]

print("a* =",a_ast)
print("b* =",b_ast)
print("La subsecuencia que debería encontrar es: [2,455,623,7,44,2,513,321,67,899,56
serie1[a_ast:b_ast+1]
```

```
[(0, 500), (1, 501), (2, 502), (3, 503), (4, 504), (5, 505), (6, 506), (7, 507), (8, 5
08), (9, 509), (10, 510), (11, 511), (12, 512), (13, 513), (14, 514), (15, 515), (16,
516), (17, 517), (18, 518), (19, 519), (20, 520), (21, 521), (22, 522), (23, 523), (24
, 524), (25, 525), (26, 526), (27, 527), (28, 528), (29, 529), (30, 530), (31, 531)]
0.0
a* = 500
b* = 531
La subsecuencia que debería encontrar es: [2,455,623,7,44,2,513,321,67,899,566,6,4,345
,44,56,567,321,4,34,456,65,67,788,8,9,432,546,563,4,232,5]

array([  2., 455., 623.,   7.,  44.,   2., 513., 321.,  67., 899., 566.,
         6.,   4., 345.,  44.,  56., 567., 321.,   4.,  34., 456.,  65.,
        67., 788.,   8.,   9., 432., 546., 563.,   4., 232.,   5.])
```

Con una colección de datos grande, encuentra el patrón a la perfección

```
serie1_2=np.concatenate((array1,serie2_menor,array2), axis=None)

path, dist = dtw_subsequence_path(serie2_menor, serie1_2)
print(path)
print(dist)

path=np.array(path)
a_ast = path[0, 1]
b_ast = path[-1, 1]

print("a* =",a_ast)
print("b* =",b_ast)
print("La subsecuencia que debería encontrar es: [2,455,623,7,44,2,546,563,4,232,5]"
serie1_2[a_ast:b_ast+1]
```

```
[(0, 500), (1, 501), (2, 502), (3, 503), (4, 504), (5, 505), (6, 506), (7, 507), (8, 5
08), (9, 509), (10, 510)]
0.0
a* = 500
b* = 510
La subsecuencia que debería encontrar es: [2,455,623,7,44,2,546,563,4,232,5]

array([  2., 455., 623.,   7.,  44.,   2., 546., 563.,   4., 232.,   5.])
```

Con una colección de datos grande y un secuencia de búsqueda algo menor, encuentra el patrón a la perfección

Figura 5.10: Ejemplo de uso de la librería *tslearn*

Obtención de las secuencias con *tslearn*

Con la función **dtw_subsequence_path** de la librería **tslearn** se calcula la medida de similitud mediante DTW entre una secuencia de referencia y otra secuencia de mayor tamaño. No es necesario que las secuencias tengan el mismo tamaño pero si que es imprescindible que tengan la misma dimensión.

tslearn.metrics.dtw_subsequence_path(subseq, longseq)

Parámetros de entrada:

1. *subseq*, serie temporal que representará el patrón que se desea encontrar.
2. *longseq*, serie temporal de mayor tamaño en la que se analizará donde se encuentra la secuencia más parecida al patrón de referencia.

Parámetros de salida:

1. Lista de pares enteros, ruta por la cual se puede encontrar la secuencia de referencia dentro de la secuencia de mayor tamaño. La lista está representada por pares de índices en los que la primera posición de cada tupla corresponden al elemento de *subseq* y el segundo elemento de la tupla a *longseq*.
2. *float*, puntuación de similitud.

En la figura 5.11 se puede observar un ejemplo de uso de esta función.

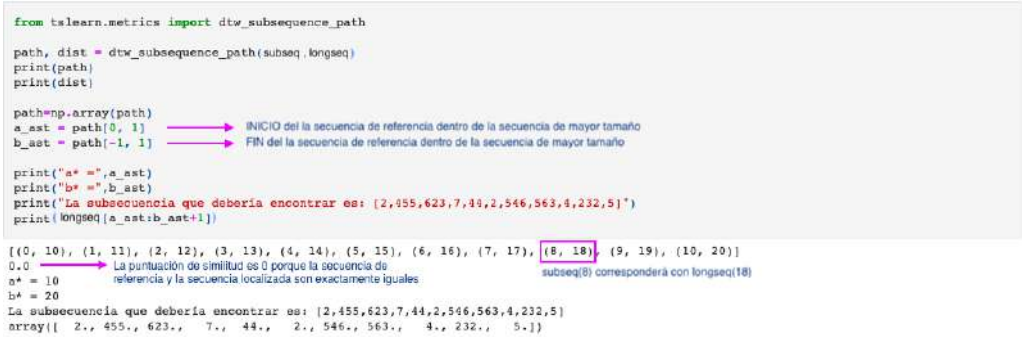


Figura 5.11: Ejemplo de uso de la función *dtw_subsequence_path*.

Finalmente matizar que el algoritmo utilizado con **tslearn** funciona tanto para valores unidimensionales como multidimensionales. En el proyecto se han realizado distintas pruebas y en algunas de ellas se han redimensionado los valores haciendo que posean una única dimensión. Esto se ha realizado por la eficiencia con la que se obtenían los resultados.

5.9. Segunda fase: Búsqueda de varias secuencias

Una vez expuesto el proceso por el cual se encuentra una secuencia dentro de otra de mayor tamaño lo más parecida posible a un patrón de referencia, ahora se va a exponer como poder almacenar varias secuencias parecidas al patrón de referencia.

Esta motivación surge de la idea de localizar múltiples secuencias para posteriormente ser analizadas y almacenar la mejor, es decir, la más parecida al patrón de referencia. Cabe destacar que las librerías anteriormente usadas hacen este proceso automáticamente, es decir, evalúan todos los posibles caminos y devuelven el que mejor distancia DTW arroje. Ahora se va a plantear obtener todas las secuencias y más adelante que sea el usuario sea el que seleccione los criterios de clasificación que mejor le convengan para obtener la secuencia deseada.

Para poder observar ejemplos de como obtener dichas secuencias se han creado un par de *notebooks*, **C4_Búsqueda_multiples_secuencias** y **C3_Búsqueda_y_agrupación_de_secuencias_SIMILARES**. En ellos se puede observar como una vez extraídas las secuencias lo primero que se realiza es calcular las matrices de coste y las posibles rutas que sirven para encontrar las secuencias más similares al patrón de referencia. Si se quiere profundizar sobre como se ha realizado el cálculo de las matrices, por favor visitar el apartado 3.9.

Como se comenta en el apartado anteriormente mencionado, a la hora de calcular la matriz de costes locales o matriz de distancias, se pueden usar diversas métricas para obtener la distancia entre las dos secuencias generadas. En este proyecto, tanto esta como todas las pruebas que conlleven el cálculo manual de dicha matriz, se han realizado utilizando la distancia *euclidiana*.

Por otra parte, para localizar todos las posibles rutas de deformación se ha puesto en práctica la función **scipy.signal.find_peaks**, esta función tiene como objetivo retornar la localización de los distintos picos que se

```python
from scipy.signal import find_peaks
from tslearn import metrics
from tslearn.generators import random_walks
from tslearn.preprocessing import TimeSeriesScalerMeanVariance

def calcule_matrix(X,Y):
    # Matriz de distancias
    C=np.zeros((len(X),len(Y)))

    for i in range(len(X)):
        for j in range(len(Y)):
            C[i][j] = np.linalg.norm(X[i]-Y[j]) #para secuencias de varias dimensiones

    # Matriz de costes acumulados
    D=np.zeros((len(X),len(Y)))
    D[:,0]=np.cumsum(C[:,0]) #inicialización de la primera columna
    D[0,:]=C[0,:] #Inicialización de la primera fila

    for i in range(1,len(X)):
        for j in range(1,len(Y)):
            D[i][j] = C[i][j]+min(D[i-1][j],D[i][j-1],D[i-1][j-1])

    return C,D

def calcule_path(C):
    cost_func = C[-1, :]

    #Identificamos los posibles caminos
    sz=10
    potential_matches = find_peaks(-cost_func, distance=sz * 0.75, height=-50)[0]

    #Calcula las rutas óptimas a partir de cada uno de los mínimos identificados
    paths = [metrics.subsequence_path(C, match) for match in potential_matches]

    return paths
```

```python
C,D=calcule_matrix(np.array(pos_ej1), np.array(pos_ej_paciente))

paths=calcule_path(D)
print("\n Número de rutas: ",len(paths))
```

Figura 5.12: Ejemplo de uso del algoritmo que localiza múltiples secuencias.

encuentran dentro de una secuencia. De esta manera podemos encontrar las secuencias similares al patrón de referencia a través de los subconjuntos de picos que se localicen en la secuencia de mayor tamaño.

En la figura 5.12 se muestra el código implementado para la búsqueda de varias secuencias. En ella se puede observar:

1. **Función que calcula las matrices de coste**, en este caso las matrices no se obtendrán por medio de ninguna librería si no que serán creadas por la función *calcule_matrix(X, Y)*. Esta función recibe como parámetros dos secuencias, la primera secuencia corresponderá a la secuencia de ejercicios que se desea localizar y la segunda secuencia corresponderá al conjunto total de ejercicios realizados por el paciente en el que se desea encontrar la secuencia de menor tamaño.

2. **Función para calcular las posibles rutas**, como se puede observar la función **scipy.signal.find_peaks** recibe tres argumentos. El primero hace referencia a la secuencia que contiene los picos, el segundo hace referencia a la distancia horizontal mínima. Este parámetro depende de la variable *sz*, cuanto mayor sea este parámetro, mayor será la

distancia entre picos y por tanto más restrictivo será el resultado obtenido. Finalmente el último parámetro que recibe la función es la altura mínima requerida para los picos. Si se hubiese recibido una secuencia con dos elementos, el primero de ellos se interpretaría como la altura mínima y el segundo como la altura máxima que podrá alcanzar el pico

```
import time
inicio = time.time()

C=calcule_matrix(ang_ej1, ang_paciente)

valores_sz=[3,7,10,14,20,30,40,50,60,70]

for sz in valores_sz:
    paths=calcule_path(C,sz)
    print("Número de caminos óptimos para sz = "+str(sz)+": ",len(paths))

fin = time.time()
print("\nTiempo trascurrido en localizar los posibles caminos óptimos: ",fin-inicio, "segundos")
```

```
Número de caminos óptimos para sz = 3:  102
Número de caminos óptimos para sz = 7:  72
Número de caminos óptimos para sz = 10:  58
Número de caminos óptimos para sz = 14:  44
Número de caminos óptimos para sz = 20:  33
Número de caminos óptimos para sz = 30:  22
Número de caminos óptimos para sz = 40:  19
Número de caminos óptimos para sz = 50:  14
Número de caminos óptimos para sz = 60:  13
Número de caminos óptimos para sz = 70:  11

Tiempo trascurrido en localizar los posibles caminos óptimos:  36.794341802597046 segundos
```

```
print(paths[0][1],paths[1][1],paths[2][1],paths[3][1],paths[4][1],paths[5][1],paths[5][1],paths[7][1]
      ,paths[8][1],paths[9][1],paths[10][1])
```

```
(1, 104) (1, 245) (1, 615) (1, 876) (1, 1066) (1, 1352) (1, 1352) (1, 1969) (1, 2192) (1, 2528) (1, 2528)
```

Figura 5.13: Ejemplo de uso del algoritmo que localiza múltiples secuencias con ángulos.

Para poder observar los resultados obtenidos sobre un conjunto de datos reales, se han extraído los ángulos de un ejercicio concreto y de una secuencia de ejercicios que comprende dicho movimiento. Aunque anteriormente se ha comentado que los ángulos son datos unidimensionales, los conjuntos de datos que se analizarán tendrán múltiples dimensiones, ya que por cada *frame* se extraen un total de doce ángulos. Por lo tanto, teniendo en cuenta que la secuencia del ejercicio concreto comprende un total de 969 *frames* y la secuencia que contiene múltiples ejercicios comprende un total de 2896 *frames*, se tendrán un patrón de referencia de dimensiones 969x12 y una secuencia larga de 2896x12. En la figura 5.13 se observan los resultados obtenidos. Cuanto mayor es el parámetro *sz*, menor es el número de secuencias que encuentra, y también se puede observar como el tiempo de ejecución para la búsqueda de todos los posibles caminos con las distintas variaciones del parámetro *sz*, es únicamente de 36.79 segundos, un resultado muy favorable.

Finalmente, este vídeo corresponde al vídeo *Ejercicio1.mp4* que se puede encontrar en el directorio **src/pruebas/vídeos/**. Y el vídeo que comprende la secuencia de ejercicios corresponde al vídeo *VideoCompleto.mp4* que se localizan en el mismo directorio. El ejercicio concreto se debería de ubicar en los 960 primeros *frames* del vídeo que contiene todos los ejercicios, pero el resultado arrojado por el algoritmo, para *sz=70* es el siguiente:

$$(1,104)(1,245)(1,615)(1,876)(1,1066)(1,1352)(1,1352)(1,1969)(1,2192)(1,2528)(1,2528)$$

Cada una de las tuplas corresponden a la posición número uno de los distintos caminos encontrados. Dentro de cada tupla, los valores se clasifican de la siguiente manera, el primer valor corresponde a la secuencia concreta y el segundo a la secuencia de mayor tamaño, es decir, para una tupla (z, y) quiere decir que la posición z de la secuencia de menor tamaño se encuentra alineado con la posición y de la secuencia completa. Y como se puede observar en el resultado, los dos primeros valores podrían darse por válidos ya que el ejercicio debería de empezar en la posición cero, pero siempre se le dará un poco de margen o tasa de error. Sin embargo el resto de secuencias deberían de ser descartadas.

Para poder clasificar estos resultados y almacenar la secuencia que más se asimile al patón de referencia se emplearán distintas métricas de clasificación. En la tabla 5.1 se exponen algunas de las que han sido usadas en este proyecto. Cabe destacar que todas esta métricas están seleccionadas para poder clasificar datos multidimensionales.

Distancias DTW
tslearn.metrics.dtw_subsequence_path(serie1, serie2)
dtaidistance.dtw_ndim.distance(serie1, serie2)
dtaidistance.dtw_ndim.distance_fast(serie1, serie2)
Distancia euclidiana
dtaidistance.dtw_ndim.ub_euclidean(serie1, serie2)
Métrica Soft-DTW
tslearn.metrics.soft_dtw(serie1, serie2, gamma)
Coste DTW
pydtw.dtw2d(serie1, serie2)
tslearn.metrics.dtw_path_from_metric(serie1, serie2)
tslearn.metrics.dtw(serie1, serie2, global_constraint="sakoe_chiba", sakoe_chiba_radius=0.5)
tslearn.metrics.dtw(serie1, serie2, global_constraint=ïtakura", itakura_max_slope=2.)

Tabla 5.1: Tabla con las posibles métricas de clasificación.

En esta fase el objetivo principal fue el de almacenar varias secuencias, por lo que para obtener los resultados de la clasificación deberemos adentrarnos en la *tercera fase*.

5.10. Tercera fase: Búsqueda del inicio y del final

Uno de los problemas que se obtiene con el planteamiento de buscar la secuencia al completo es la longitud de la secuencia localizada. Dicha longitud será igual o menor a la secuencia pasada como referencia y esto puede ser un inconveniente.

Si un paciente realiza el mismo ejercicio que su terapeuta pero en un tiempo algo mayor se generarán las siguientes secuencias:

1. **Secuencia T**, esta secuencia corresponderá al ejercicio que realiza el terapeuta con $T = [t_1, t_2...t_N]$ siendo N la longitud de la secuencia.

2. **Secuencia P**, esta secuencia corresponderá a los múltiples ejercicios realizados por el paciente con $P = [p_1, p_2...p_M]$ siendo M la longitud de la secuencia y sabiendo que $M > N$

3. **Secuencia S**, esta secuencia corresponderá al ejercicio que se desea encontrar dentro de la secuencia P con $S = [s_1, s_2...s_K]$ siendo K la longitud de la secuencia y sabiendo que $S \in P$ y que $K > N$

Si el algoritmo únicamente localiza una secuencia dentro de otra de mayor tamaño, estará localizando la secuencia S dentro de la secuencia P con un tamaño de N en vez de localizar la secuencia con su tamaño K. Esto producirá como resultado vídeos incompletos, que o bien acaban antes de tiempo o empiezan más tarde de lo esperado.

Para abordar este problema se ha planteado la búsqueda de inicios y finales independientes. Para ello se han seguido los siguientes pasos:

1. Obtener las x primeras y últimas posiciones de cada ejercicio. Este cálculo se puede realizar mediante un porcentaje, por ejemplo el 10 % de la longitud del ejercicio, aunque en las pruebas realizadas se ha ido modificando este porcentaje para poder observar los diferentes resultados.

2. Realizar la búsqueda de varias secuencias tanto sobre el supuesto inicio como sobre el supuesto final. De esta forma se han obtenido numerosos inicio y finales, que por supuesto dependerán de los parámetros de restricción del algoritmo, en concreto del parámetro *distance* anteriormente nombrado.

3. Enlazar cada posible inicio con cada posible final. Para poder crear una ruta de deformación lo más óptima posible, se ha enlazado la posición de inicio de cada posible secuencia de inicio con la posición de final de cada posible secuencia de final, es decir, si contamos con una ruta de inicio que comienza en el *frame* 200 y finaliza en el 400, y por otra parte, dos secuencias de final que comienzan en los *frames* 600, y 700 y finalizan en los 1300, 1400. Se crearán dos rutas de deformación, la primera abarcará desde el *frame* 200 hasta el 1300 y la segunda desde el 200 al 1400.

4. Una vez obtenidas todas las posibles rutas de deformación, el último paso es procesar cada una de las secuencias generadas y aplicarle ciertos criterios para que sean desechadas o aceptadas. Uno de los principales criterios de clasificación es el tamaño. Si la secuencia obtenida tiene un tamaño similar al de la secuencia de referencia con una cierta tasa de error, se aceptará como posible solución. Más adelante se aplicarán las distancias DTW de la secuencia localizada con respecto de la secuencia de referencia. Si las distancias tienen un valor demasiado elevado quiere decir que las secuencias no se están alineando correctamente y por lo tanto que no tienen valores similares. Si por el contrario estas distancias arrojan valores bajos estará expresando que las secuencias son bastante parecidas y por tanto que se ha encontrado una secuencia óptima.

Una vez se conoce el proceso por el cual se empiezan a clasificar las secuencias, la duda que surge ahora es ¿cómo saber cuál es la mejor secuencia? en otras palabras, ¿cómo saber cuál es la secuencia más parecida a un patrón de referencia?.

Como se ha comentado anteriormente el tamaño es un factor clave. Muchas veces las secuencias que se obtienen son demasiado parecidas, es decir podemos tener almacenados inicios o finales consecutivos o simplemente valores demasiado cercanos. Al aplicar la unión entre los posibles inicios y los posibles finales nos pueden quedar secuencias demasiado pequeñas, o en otros casos demasiado grandes. Por ejemplo si tenemos las siguientes secuencias:

1. secuencia-ejercicio1, con una longitud de 50 posiciones.

2. secuencia-ejercicio-completo, con una longitud de 50 posiciones.

3. *posibles_inicios*, [1,3,12,33,41,100].

4. *posibles_finales*, [2,35,65,150,187].

Al juntar cada posible inicio con cada posible final, algunas de las posibles rutas que se calcularán son: secuencia_ejercicio_completo[1-2], secuencia_ejercicio_completo[1-35], secuencia_ejercicio_completo[1-65], secuencia_ejercicio_completo[1-150]. A simple vista se puede observar como la primera secuencia es demasiado pequeña y la última demasiado grande. Las otras dos secuencias no son perfectas pero son aceptables.

Es por esta razón que el primer filtro a realizar es quedarse únicamente con las secuencias que sean similares al patrón de referencia. Una vez localizadas dichas secuencias el siguiente paso será buscar el grado de similitud que tienen con respecto del patrón de referencia. Para ello se usarán las técnicas investigadas en la **segunda fase**.

Inicio	Fin	Distancia DTW con tsleran	Distancia DTW con dtaidistance	Coste con pydtw
49	229	2015.7734436854455	2242.0713006630585	3075767.82365
123	329	1974.9796179538437	2277.492815798004	3623770.971422
221	391	2013.5418212417126	2173.8919777361834	2903770.599562
245	420	2016.4642688386466	2203.8221787693333	inf
289	515	1967.3959282959777	2378.808525040852	3947960.183013
351	547	1927.2893178181143	2281.1751682487525	3531987.928479
396	572	1908.5767256769245	2173.755612732177	inf

Tabla 5.2: Análisis de los datos obtenidos con inicios y finales.

En la tabla 5.2 se pueden observar algunos los datos obtenidos de una ejecución en la que se han localizado un total de 16 posibles inicios óptimos y 13 posibles finales. Las diferencias de distancia y coste entre las posibles secuencias son demasiado parecidas como para poder hacer una clasificación con ellas.

Este ha sido el problema fundamental por el que se ha descartado esta posible solución. Tras probar con múltiples librerías y funciones se llegó a la conclusión de que no se podían clasificar correctamente los ejercicios usando estas métricas por la similitud en sus resultados.

Finalmente comentar que tras los resultados anteriormente mostrados se realizaron múltiples pruebas sobre diferentes algoritmos que calculasen distancias de otro tipo como la distancia *euclidiana* o la *manhattan*. Nuevamente los resultados no sirvieron para poder obtener una buena clasificación de las secuencias.

5.11. Cuarta fase: Búsqueda de una secuencia intermedia

Esta fase surge de la misma línea de pensamiento que la fase anterior, pero en este caso lo que se planteo es buscar una secuencia que representase el punto medio del patrón de referencia. Esta idea surge por la similitud de movimientos en los inicios y finales de los ejercicios. Como posible solución se planteo la búsqueda de esta secuencia intermedia y seguidamente añadirla una cierta longitud extra tanto previa como posteriormente.

El problema de este planteamiento fueron varios. Por una parte localizaba multitud de rutas de deformación, lo que conllevaba a múltiples secuencias intermedias. Una vez obtenidas estas secuencias, se puso nuevamente esperanza en el método de clasificación empleado en la **tercera fase**, pero los problemas acabaron siendo los mismos. Al igual que los resultados mostrados en la tabla 5.2, en este caso tampoco se pudo realizar una buena clasificación.

Por otra parte, aunque algunos ejercicios lograsen ser bien clasificados, el problema se encontraba al aplicar la tasa, es decir, al añadirle una longitud extra tanto por la parte inmediatamente anterior como posterior. Muchos de los ejercicios siguen un mismo patrón, por ejemplo, levantar alternativamente las manos un cierto número de veces. Si el algoritmo encontraba como camino óptimo la secuencia inicial, pensándose que es la intermedia, tras añadir a dicha secuencia las posiciones que le faltaban, creaba un ejercicio tomando parte de los anteriores.

Tanto en esta fase como en la anterior se realizaron múltiples pruebas cambiando el tamaño de las secuencias de inicio, fin e intermedias, pero con esta técnica simplemente se obtenían un mayor número de posibles resultados y a la hora de clasificarlos surgían nuevamente los mismos problemas.

Por estas razones estos planteamientos fueron descartados tras no obtener resultados concluyentes.

5.12. Quinta fase: Búsqueda de la secuencia completa

En esta fase se ha procedido a localizar la secuencia completa del ejercicio concreto dentro de la secuencia de mayor tamaño. Para ello esta **quinta fase** se ha dividido dos *subfases*, como se puede observar en la figura 5.14, por una parte se ha procedido a analizar las secuencias con datos multidimensionales,

y tras ello se redimensionaron las secuencias para convertirlas en unidimensionales. En los *notebooks* **C6_Pruebas_busqueda_angulos.ipynb** y **C7_Pruebas_busqueda_posiciones.ipynb** se pueden observar al detalle las distintas implementaciones.

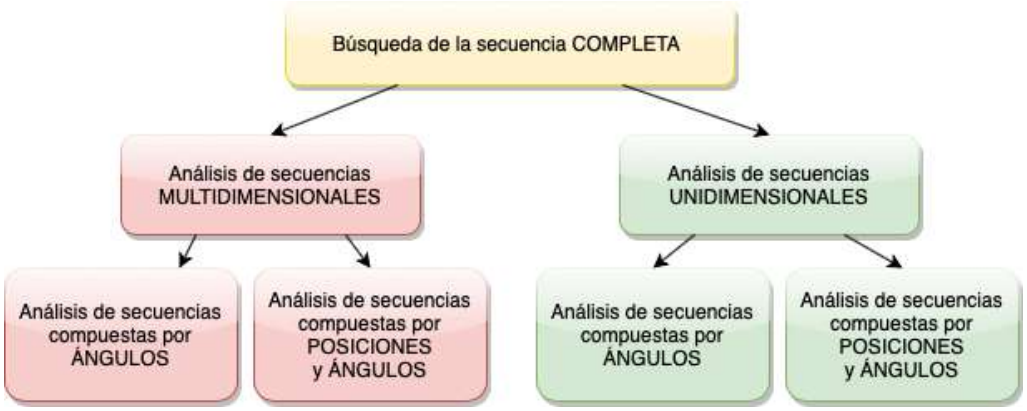


Figura 5.14: Diagrama de la quinta fase.

Análisis de secuencias multidimensionales

Como se ha comentado en el apartado 4.8 la librería **tslearn** sirve tanto para secuencias unidimensionales como multidimensionales. Es por esta razón que será usada para todos los tipos de secuencias propuestas.

Esta métrica de realizar las comprobaciones con secuencias multidimensionales arrojaba unos tiempos de ejecución estupendos, como se puede observar en la tabla 5.3. Esta tabla ha sido creada a partir de los tiempos de ejecución de cuatro secuencias compuestas únicamente por ángulos, aunque cabe destacar que con conjuntos de datos compuestos por posiciones y ángulos, los resultados son prácticamente iguales. Pero el coste de unos tiempos tan buenos son unos resultados en cuanto a la localización de secuencias nefastos. Como se puede observar en la imagen 5.15 los resultados arrojados son excesivamente malos. Incluso después de eliminar los valores nulos, encuentra una secuencia de ejercicios con la friolera duración de dos *frames*, en la que el ejercicio supuestamente comienza en la posición 1853 y termina en la posición 1856. Este tipo de resultados son totalmente inviables y tras la realización de numerosas pruebas tanto con ángulos como con la combinación de estos mismos con el resto de posiciones, se llegó a la conclusión de que trabajar con datos multidimensionales no era una opción. Aunque los tiempos de ejecución sean estupendos, no se pueden permitir unos resultados que se alejan tanto de la solución esperada.

Ejercicio	Frames	Tiempo de procesamiento
Ejercicio 1	969	0.0743436 segundos
Ejercicio 2	624	0.0438256 segundos
Ejercicio 3	626	0.0455613 segundos
Ejercicio 4	439	0.0343325 segundos

Tabla 5.3: Tiempos de ejecución sobre una secuencia multidimensional larga de **2897 frames**

Búsqueda del segundo ejercicio

```
inicio = time.time()

path, dist = dtw_subsequence_path(array_angulos2, array_angulos_totales)

path=np.array(path)
a_ast = path[0, 1]
b_ast = path[-1, 1]

fin = time.time()

print("El Ejercicio 2 comienza en el frame =",a_ast)
print("El Ejercicio 2 finaliza en el frame =",b_ast)
print("\nEl tiempo transcurido en esta búsqueda ha sido de: ",fin-inicio," segundos")
```

```
El Ejercicio 2 comienza en el frame = 1853
El Ejercicio 2 finaliza en el frame = 1856

El tiempo transcurido en esta búsqueda ha sido de:  0.043825626373291016  segundos
```

En este caso la secuencia que encuentra es ridículamente pequeña y no se ubica dentro de las posibles soluciones ya que detecta la secuencia de inicio después de la secuencia de fin real.

Búsqueda del tercer ejercicio

```
inicio = time.time()

path, dist = dtw_subsequence_path(array_angulos3, array_angulos_totales)

path=np.array(path)
a_ast = path[0, 1]
b_ast = path[-1, 1]

fin = time.time()

print("El Ejercicio 3 comienza en el frame =",a_ast)
print("El Ejercicio 3 finaliza en el frame =",b_ast)
print("\nEl tiempo transcurido en esta búsqueda ha sido de: ",fin-inicio," segundos")
```

```
El Ejercicio 3 comienza en el frame = 357
El Ejercicio 3 finaliza en el frame = 643

El tiempo transcurido en esta búsqueda ha sido de:  0.04556131362915039  segundos
```

Nuevamente encuentra una secuencia demasiado pequeña y en posiciones incorrectas

Figura 5.15: Pruebas sobre secuencias multidimensionales compuestas por ángulos.

Análisis de secuencias unidimensionales

Al igual que en el caso de las secuencias multidimensionales, en esta ocasión los resultados también fueron obtenidos a través del análisis que ofrece la librería **tslearn**.

Ejercicio	Frames	Tiempo de procesamiento
Ejercicio 1	969	4.927449 segundos
Ejercicio 2	624	2.457397 segundos
Ejercicio 3	626	2.4901232 segundos
Ejercicio 4	439	1.7612924 segundos

Tabla 5.4: Tiempos de ejecución sobre una secuencia unidimensional larga de **2897 frames**

En la tabla 5.4 se pueden observar los tiempos de ejecución de los mismos ejercicios que en la tabla 5.3 pero en este caso las secuencias se han redimensionado a una única dimensión. Como se puede apreciar, los tiempos son algo mayores pero muy aceptables, sobretodo después de comprobar la diferencia en cuanto al resultado obtenido.

Como se puede apreciar en la figura 5.16 la secuencia correspondiente al segundo ejercicio es localizada a la perfección mientras que la correspondiente al tercero no lo es. De la misma forma y como se puede apreciar en el *notebook* **C6_Pruebas_busqueda_angulos.ipynb, C7_Pruebas_busqueda_posiciones.ipynb** la secuencia correspondiente al cuarto ejercicio también está correctamente localizada mientras que la relativa al primer ejercicio no. Por lo tanto, hasta este punto se ha conseguido un 50 % de éxito que ya es una diferencia considerable con las técnicas anteriormente planteadas que no conseguían resultados nada favorables.

Tras comprobar que el análisis de secuencias unidimensionales con conjuntos de datos compuestos únicamente por ángulos arrojaba unos resultados bastante favorables, se probó a realizar la misma técnica pero en este caso con secuencias que almacenaban tanto los ángulos como el resto de posiciones del esqueleto.

Como se puede comprobar en la tabla 5.5 los tiempos de ejecución en este caso son bastante mayores, pero aún así aceptables y los resultados obtenidos fueron peores, clasificando de forma correcta únicamente el primer ejercicio.

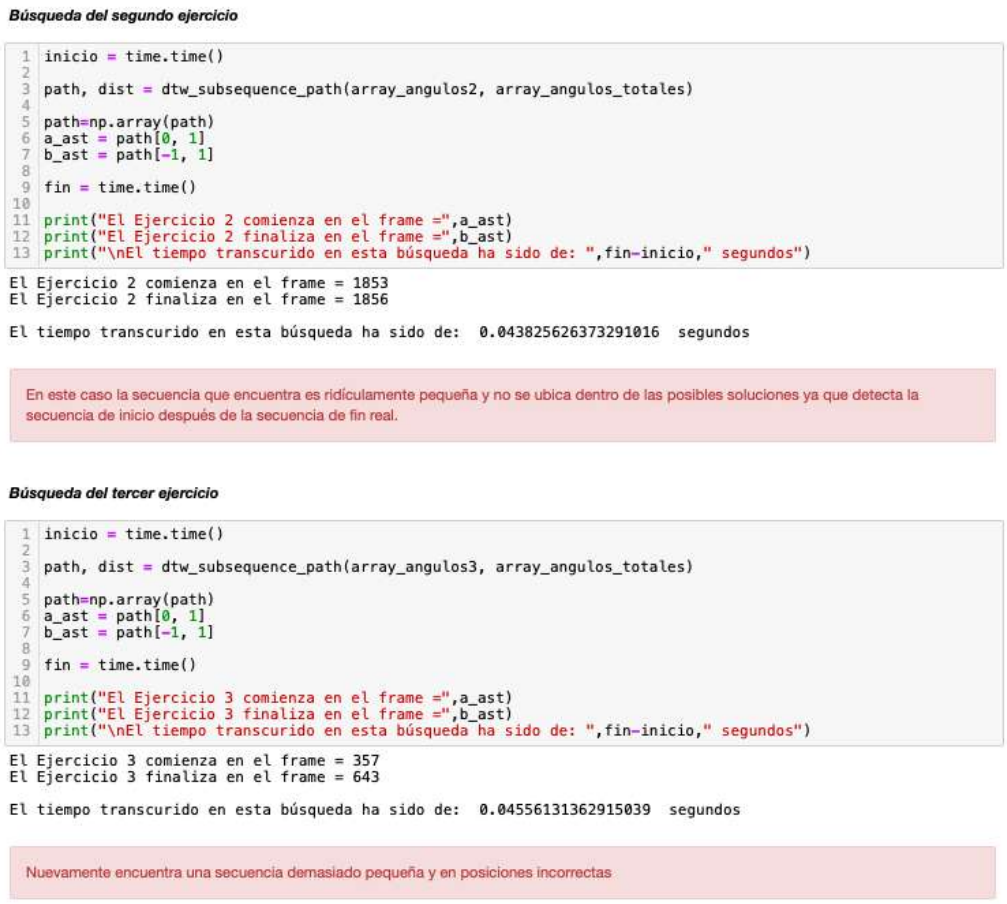


Figura 5.16: Pruebas sobre secuencias unidimensionales compuestas por ángulos.

Ejercicio	Frames	Tiempo de procesamiento
Ejercicio 1	969	76.11154 segundos
Ejercicio 2	624	48.85858 segundos
Ejercicio 3	626	48.74371 segundos
Ejercicio 4	439	34.54078 segundos

Tabla 5.5: Tiempos de ejecución sobre una secuencia unidimensional larga de **2897 frames** compuesta por todas las posiciones y ángulos.

Con los resultados obtenidos se decidió plantear una nueva posible solución. Los ejercicios que se pueden observar en los diferentes vídeos son ejercicios lentos que repiten un patrón, pensados para personas con la enfermedad de *Parkinson*. Estas personas son, por lo general, personas

mayores a las que les cuesta realizar movimientos y por lo tanto las secuencias de posiciones generadas a partir de sus vídeos comprenderán datos muy similares entre si. Es por esta razón por la que se decidió probar el análisis de las secuencias recortando algunos *frames*.

Recortando algunos *frames* también se conseguiría disminuir los tiempos de ejecución y mejorar los resultados. Para poder obtener unos resultados óptimos se han realizado numerosas pruebas con distinto número de *frames* pero en todas y cada una de las pruebas se seguían los siguientes pasos:

1. **Redimensionar los datos**, haciendo que las dimensiones sean unidimensionales.
2. **Eliminar los valores nulos**, para que encuentre una secuencia óptima es muy importante deshacerse de los valores nulos.
3. **Recortar frames**, en cada uno de las ejecuciones se iban recortando un poco más y analizando los resultados obtenidos.
4. **Calcular su correspondencia respecto al *frame* original**, para comprobar si el resultado obtenido es correcto, se debe aplicar una sencilla regla de tres y comprobar si corresponde con el *frame* real.

Tras la creación de la aplicación de escritorio, que se comenta en el apartado 5.15, la comprobación de resultados fue fantástica, ya que una vez obtenidos los *frames* de inicio y de final, se introducen los valores en ella y se puede ver lo bien o mal que está localizando la secuencia.

Como conclusión final a este apartado exponer que los mejores datos han sido obtenidos a partir del aproximadamente 70 % de los *frames* y se pueden observar los resultados en el cuaderno **C10_Resultados_finales**.

5.13. Clasificación de ejercicios en las extremidades superiores e inferiores

Para clasificar ejercicios según la extremidad del cuerpo que esté en movimiento se ha planteado una posible solución. Esta clasificación se consigue aplicando la desviación típica sobre el conjunto de datos obtenido.

La desviación típica o estándar cuantifica la variación o la dispersión del conjunto de datos a analizar. Una desviación estándar baja indica que una gran parte de los datos procesados tienden a estar agrupados y a no

sufrir grandes diferencias entre ellos, mientras que una desviación estándar elevada indica que los datos se extienden sobre un rango de valores bastante amplio.

Este concepto se puede aplicar en el proyecto de la siguiente manera, si un individuo realiza movimientos sobre las extremidades superiores, éstas constantemente estarán en posiciones distintas lo que creará un conjunto de datos bastante disperso. Mientras, en ese mismo ejercicio, las extremidades inferiores estarán prácticamente inmóviles creando un conjunto de datos muy similar, es decir, un conjunto de datos muy agrupado.

Para conseguir una clasificación óptima se han probado varias técnicas:

1. Probar la desviación típica sobre todo el conjunto de datos.

2. Probar la desviación típica sobre el eje x y el eje y.

3. probar la desviación típica sobre distintas partes del cuerpo.

Para ejecutar las pruebas se han separado los datos en dos conjuntos, por una parte se encuentran los valores asociados a las partes superiores del cuerpo y por otra los valores asociados a la parte inferior. Como partes superiores se entienden, nariz, hombros, cuello ángulos del cuello, codos, manos, ángulos del codo, y ángulos del hombro. Y finalmente como partes inferiores se han escogido la cadera, rodillas, ángulos tanto de la cadera como de las rodillas y los tobillos.

Con los datos separados se ha calculado la desviación estándar dos veces sobre el mismo ejercicio. Una vez sobre los datos que corresponden a las posiciones superiores del esqueleto y una segunda vez sobre los datos de las posiciones inferiores. Una vez obtenidos los dos resultados se compararán y supuestamente el que tenga mayor desviación estándar será el que indique el tipo de ejercicio que se está realizando. Se precisa de la palabra supuestamente porque a continuación se van a exponer los distintos planteamientos y las posibles soluciones que se han implementado.

Finalmente, para comprender los ejemplos se van a exponer los ejercicios analizados en la tabla 5.6 y el tipo de movimiento que se realiza.

Ejercicio realizado	Parte sobre la que se realiza
Ejercicio 1	Extremidades SUPERIORES
Ejercicio 2	Extremidades INFERIORES
Ejercicio 3	Extremidades SUPERIORES
Ejercicio 4	Extremidades SUPERIORES
Ejercicio 5	Extremidades INFERIORES
Ejercicio 6	Extremidades INFERIORES
Ejercicio 7	Extremidades INFERIORES
Ejercicio 8	Extremidades INFERIORES
Ejercicio 9	Extremidades SUPERIORES
Ejercicio 10	Extremidades SUPERIORES
Ejercicio 11	Extremidades SUPERIORES
Ejercicio 12	Extremidades SUPERIORES
Ejercicio 13	Extremidades SUPERIORES
Ejercicio 14	Extremidades SUPERIORES

Tabla 5.6: Clasificación de los ejercicios a analizar.

Pruebas sobre todo el conjunto de datos

Al probar la desviación estándar sobre todo el conjunto de datos los resultados han sido muy desalentadores. En la tabla 5.7 se muestran algunos de los datos obtenidos.

Ejercicio realizado	Desviación típica del ejercicio
Parte superior del Ejercicio 1	312.55
Parte inferior del Ejercicio 1	510.21
Parte superior del Ejercicio 2	317.73
Parte inferior del Ejercicio 2	478.92
Parte superior del Ejercicio 3	350.69
Parte inferior del Ejercicio 3	497.06
Parte superior del Ejercicio 4	313.89
Parte inferior del Ejercicio 4	525.45
Parte superior del Ejercicio 5	316.19
Parte inferior del Ejercicio 5	397.67
Parte superior del Ejercicio 6	295.88
Parte inferior del Ejercicio 6	463.63

Tabla 5.7: Desviación típica sobre cada ejercicio.

Como se puede observar, con los datos arrojados no se puede realizar ninguna clasificación. En todos los casos la desviación estándar menor se encuentra localizada en los ejercicios de la parte superior del cuerpo por lo que según los resultados presentes, todos los ejercicios deberían ser movimientos sobre las extremidades inferiores.

Al realizar el análisis de esta manera se están juntando los valores de las dimensiones de cada posición del esqueleto, esta diferencia entre las posiciones hace que los datos sean muy dispersos. En la tabla 5.8 se muestran algunos valores que toma la cadera, tanto en el eje X como en el eje Y.

Frame escogido	**Eje X**	**Eje Y**
Frame 1	453.41833	1030.679
Frame 2	458.34384	1056.794
Frame 3	459.32187	1063.948
Frame 4	460.7943	1056.463
Frame 5	454.8593	983.5553
Frame 6	455.0474	1048.695
Frame 7	450.5780	1032.291
Frame 8	451.0298	1028.209

Tabla 5.8: Valores de la desviación típica sobre la cadera.

Si se analizan por separado los valores del eje X y del eje Y se puede observar que están bastante agrupados pero que al juntar ambas dimensiones los valores quedan bastante dispersos. Este es un ejemplo de los ocho primeros esqueletos que está generando el algoritmo y es sólo para mostrar la dispersión entre las dimensiones. A lo largo de las ejecuciones los valores van alternándose por diferentes motivos como puede ser un cambio de postura o un movimiento involuntario.

Pruebas sobre el eje X y el eje Y

Para evitar la dispersión que genera la agrupación de dimensiones se ha optado por separarlas y calcular la desviación típica de cada ejercicio cuatro veces, se calcula las posiciones superiores en el eje X y en el eje Y, y finalmente las posiciones inferiores se someten al mismo procedimiento. En la tabla 5.9 se pueden observar las desviaciones estándar sobre cada ejercicio y cada eje.

Ejercicio realizado	Eje X	Eje Y
Parte superior del Ejercicio 1	219.23	382.54
Parte inferior del Ejercicio 1	170.64	655.39
Parte superior del Ejercicio 2	198.50	400.81
Parte inferior del Ejercicio 2	172.03	616.69
Parte superior del Ejercicio 3	217.49	441.42
Parte inferior del Ejercicio 3	163.35	638.46
Parte superior del Ejercicio 4	205.92	389.40
Parte inferior del Ejercicio 4	169.70	671.78
Parte superior del Ejercicio 5	218.98	389.03
Parte inferior del Ejercicio 5	190.47	512.78
Parte superior del Ejercicio 6	202.11	365.93
Parte inferior del Ejercicio 6	160.63	600.16

Tabla 5.9: Desviación típica sobre todo el conjunto de datos

Nuevamente, con los datos obtenidos no se puede realizar ninguna clasificación. En el eje X se obtiene una desviación típica mayor en todos los ejercicios que se realizan con la parte superior del cuerpo, mientras que en los resultados obtenidos en el eje Y identifica una desviación típica mayor en los ejercicios que se realizan sobre las partes inferiores del cuerpo. Sobre ninguna de estas clasificaciones podemos sacar alguna conclusión válida.

La razón de que fallase no es únicamente de las dimensiones. Es correcto separar las dimensiones por lo explicado anteriormente pero siguiendo en esa misma línea hay que indagar un poco más sobre los resultados. En este caso se están comparando todas las posiciones de los distintos ejes. Con todas las posiciones estamos comparando tanto rodillas, como tobillos como caderas. Esto quiere decir que aunque a lo largo de las ejecuciones no se haya apreciado prácticamente movimiento en las posiciones analizadas y la combinación de los diferentes puntos no varíe demasiado, entre ellos varía lo suficiente como para que la desviación típica arroje un resultado desfavorable.

Al analizar la figura 3.4 se puede observar que donde más varían las posiciones es a lo largo del eje Y, en concreto, en las partes inferiores del esqueleto, la diferencia entre la cadera y el tobillo en relación al eje Y es bastante más notable que la posición del hombro respecto de las manos. Por esta razón las desviaciones típicas del eje Y son bastante mayores que las del eje X

Con los resultados mostrados lo más preciso parece ser analizar cada punto del esqueleto.

Pruebas sobre distintas partes del esqueleto

Después de observar los inconvenientes que genera agrupar las distintas posiciones del esqueleto, se va a proceder a evaluar individualmente algunos de los puntos relevantes que extrae el esqueleto. En las figuras 5.17 5.18 5.19 se muestran los resultados obtenidos tras calcular la desviación típica sobre los distintos ejes de cada ejercicio y sobre una posición concreta dentro de cada ejercicio.

Finalmente se puede concluir que con los resultados obtenidos si que se puede hacer una muy buena clasificación de si los ejercicios se realizan en la parte superior o inferior del cuerpo. Además plantearlo de esta manera es muy interesante para detectar en un futuro si el ejercicio se realiza sobre una parte más concreta del cuerpo como pueden ser movimientos de cuello.

Según los datos observados en las figuras 5.17 5.18 5.19 las mejores clasificaciones se han obtenido en las posiciones del eje Y de algunos valores, pero sin duda, los mejores resultados provienen del análisis de ciertos ángulos.

Hay que tener en cuenta que aunque los pacientes realicen ejercicios sobre las extremidades superiores, en algún momento ejercerán un movimiento involuntario sobre alguna extremidad inferior como puede ser un cambio en la posición o un movimiento involuntario del pie. Es por esta razón que se debe intentar encontrar la mejor agrupación para el análisis de datos

```
Posicion de la mano derecha en el eje y dentro del Ejercicio 1:  296.5888
Posicion de la mano derecha en el eje y dentro del Ejercicio 2:  33.02471
Posicion de la mano derecha en el eje y dentro del Ejercicio 3:  399.5404
Posicion de la mano derecha en el eje y dentro del Ejercicio 4:  120.49732
Posicion de la mano derecha en el eje y dentro del Ejercicio 5:  7.999684
Posicion de la mano derecha en el eje y dentro del Ejercicio 6:  12.561245
Posicion de la mano derecha en el eje y dentro del Ejercicio 7:  9.672473
Posicion de la mano derecha en el eje y dentro del Ejercicio 8:  18.610743
Posicion de la mano derecha en el eje y dentro del Ejercicio 9:  231.05264
Posicion de la mano derecha en el eje y dentro del Ejercicio 10:  297.32397
Posicion de la mano derecha en el eje y dentro del Ejercicio 11:  420.4826
Posicion de la mano derecha en el eje y dentro del Ejercicio 12:  124.268166
Posicion de la mano derecha en el eje y dentro del Ejercicio 13:  59.043007
Posicion de la mano derecha en el eje y dentro del Ejercicio 14:  53.632755
```

(a) Desviaciones típicas en la mano derecha.

```
Posicion ángulo del codo izquierdo dentro del Ejercicio 1:  83.47592334942935
Posicion ángulo del codo izquierdo dentro del Ejercicio 2:  16.18230339167514
Posicion ángulo del codo izquierdo dentro del Ejercicio 3:  67.77218096733516
Posicion ángulo del codo izquierdo dentro del Ejercicio 4:  118.05888885197976
Posicion ángulo del codo izquierdo dentro del Ejercicio 5:  3.059813112404422
Posicion ángulo del codo izquierdo dentro del Ejercicio 6:  3.4066411851481053
Posicion ángulo del codo izquierdo dentro del Ejercicio 7:  3.0557557805129805
Posicion ángulo del codo izquierdo dentro del Ejercicio 8:  2.0673297376247235
Posicion ángulo del codo izquierdo dentro del Ejercicio 9:  51.359514989789325
Posicion ángulo del codo izquierdo dentro del Ejercicio 10:  36.50330163158551
Posicion ángulo del codo izquierdo dentro del Ejercicio 11:  51.0428771020675
Posicion ángulo del codo izquierdo dentro del Ejercicio 12:  68.19544114893299
Posicion ángulo del codo izquierdo dentro del Ejercicio 13:  110.85303467627587
Posicion ángulo del codo izquierdo dentro del Ejercicio 14:  112.99166128086776
```

(b) Desviaciones típicas en el ángulo del codo izquierdo.

```
Posicion ángulo del codo derecho dentro del Ejercicio  1:  85.25224181528428
Posicion ángulo del codo derecho dentro del Ejercicio  2:  15.125709453717702
Posicion ángulo del codo derecho dentro del Ejercicio  3:  71.21521309394882
Posicion ángulo del codo derecho dentro del Ejercicio  4:  116.98337624451523
Posicion ángulo del codo derecho dentro del Ejercicio  5:  2.0345184104188867
Posicion ángulo del codo derecho dentro del Ejercicio  6:  3.9704060960416294
Posicion ángulo del codo derecho dentro del Ejercicio  7:  1.953338934357329
Posicion ángulo del codo derecho dentro del Ejercicio  8:  2.0157281436655436
Posicion ángulo del codo derecho dentro del Ejercicio  9:  61.52601612427234
Posicion ángulo del codo derecho dentro del Ejercicio  10:  38.66153236368584
Posicion ángulo del codo derecho dentro del Ejercicio  11:  54.72271065495049
Posicion ángulo del codo derecho dentro del Ejercicio  12:  68.66923612687557
Posicion ángulo del codo derecho dentro del Ejercicio  13:  116.35419281901424
Posicion ángulo del codo derecho dentro del Ejercicio  14:  118.2960634108614
```

(c) Desviaciones típicas en el ángulo del codo derecho.

Figura 5.17: Pruebas sobre diferentes desviaciones típicas.

```
Posicion de la rodilla izquierda en el eje y dentro del Ejercicio 1:  16.040947
Posicion de la rodilla izquierda en el eje y dentro del Ejercicio 2:  104.72785
Posicion de la rodilla izquierda en el eje y dentro del Ejercicio 3:  23.911211
Posicion de la rodilla izquierda en el eje y dentro del Ejercicio 4:  14.701019
Posicion de la rodilla izquierda en el eje y dentro del Ejercicio 5:  81.23107
Posicion de la rodilla izquierda en el eje y dentro del Ejercicio 6:  138.62482
Posicion de la rodilla izquierda en el eje y dentro del Ejercicio 7:  96.47382
Posicion de la rodilla izquierda en el eje y dentro del Ejercicio 8:  109.68346
Posicion de la rodilla izquierda en el eje y dentro del Ejercicio 9:  16.373013
Posicion de la rodilla izquierda en el eje y dentro del Ejercicio 10:  10.783459
Posicion de la rodilla izquierda en el eje y dentro del Ejercicio 11:  23.841154
Posicion de la rodilla izquierda en el eje y dentro del Ejercicio 12:  19.81529
Posicion de la rodilla izquierda en el eje y dentro del Ejercicio 13:  25.554098
Posicion de la rodilla izquierda en el eje y dentro del Ejercicio 14:  25.877493
```

(a) Desviaciones típicas la rodilla izquierda.

```
Posicion de la rodilla derecha en el eje y dentro del Ejercicio 1:  14.992826
Posicion de la rodilla derecha en el eje y dentro del Ejercicio 2:  105.745804
Posicion de la rodilla derecha en el eje y dentro del Ejercicio 3:  25.193604
Posicion de la rodilla derecha en el eje y dentro del Ejercicio 4:  15.23253
Posicion de la rodilla derecha en el eje y dentro del Ejercicio 5:  71.477486
Posicion de la rodilla derecha en el eje y dentro del Ejercicio 6:  134.29553
Posicion de la rodilla derecha en el eje y dentro del Ejercicio 7:  97.012566
Posicion de la rodilla derecha en el eje y dentro del Ejercicio 8:  97.086334
Posicion de la rodilla derecha en el eje y dentro del Ejercicio 9:  13.709168
Posicion de la rodilla derecha en el eje y dentro del Ejercicio 10:  11.662367
Posicion de la rodilla derecha en el eje y dentro del Ejercicio 11:  23.765797
Posicion de la rodilla derecha en el eje y dentro del Ejercicio 12:  17.218292
Posicion de la rodilla derecha en el eje y dentro del Ejercicio 13:  25.337639
Posicion de la rodilla derecha en el eje y dentro del Ejercicio 14:  26.962223
```

(b) Desviaciones típicas en la rodilla derecha.

```
Posicion del tobillo izquierdo en el eje y dentro del Ejercicio 1:  17.254868
Posicion del tobillo izquierdo en el eje y dentro del Ejercicio 2:  92.78449
Posicion del tobillo izquierdo en el eje y dentro del Ejercicio 3:  25.000359
Posicion del tobillo izquierdo en el eje y dentro del Ejercicio 4:  16.270868
Posicion del tobillo izquierdo en el eje y dentro del Ejercicio 5:  124.64031
Posicion del tobillo izquierdo en el eje y dentro del Ejercicio 6:  125.269585
Posicion del tobillo izquierdo en el eje y dentro del Ejercicio 7:  88.11312
Posicion del tobillo izquierdo en el eje y dentro del Ejercicio 8:  80.95424
Posicion del tobillo izquierdo en el eje y dentro del Ejercicio 9:  12.105806
Posicion del tobillo izquierdo en el eje y dentro del Ejercicio 10:  9.664043
Posicion del tobillo izquierdo en el eje y dentro del Ejercicio 11:  17.74969
Posicion del tobillo izquierdo en el eje y dentro del Ejercicio 12:  19.416697
Posicion del tobillo izquierdo en el eje y dentro del Ejercicio 13:  27.31135
Posicion del tobillo izquierdo en el eje y dentro del Ejercicio 14:  27.287348
```

(c) Desviaciones típicas en el tobillo izquierdo.

Figura 5.18: Pruebas sobre diferentes desviaciones típicas.

```
Posicion del tobillo derecho en el eje y dentro del Ejercicio 1:  15.228322
Posicion del tobillo derecho en el eje y dentro del Ejercicio 2:  85.59574
Posicion del tobillo derecho en el eje y dentro del Ejercicio 3:  26.175419
Posicion del tobillo derecho en el eje y dentro del Ejercicio 4:  16.078903
Posicion del tobillo derecho en el eje y dentro del Ejercicio 5:  118.11835
Posicion del tobillo derecho en el eje y dentro del Ejercicio 6:  122.4812
Posicion del tobillo derecho en el eje y dentro del Ejercicio 7:  89.59549
Posicion del tobillo derecho en el eje y dentro del Ejercicio 8:  63.41441
Posicion del tobillo derecho en el eje y dentro del Ejercicio 9:  11.141483
Posicion del tobillo derecho en el eje y dentro del Ejercicio 10:  12.131502
Posicion del tobillo derecho en el eje y dentro del Ejercicio 11:  18.662254
Posicion del tobillo derecho en el eje y dentro del Ejercicio 12:  16.846992
Posicion del tobillo derecho en el eje y dentro del Ejercicio 13:  26.8382
Posicion del tobillo derecho en el eje y dentro del Ejercicio 14:  27.323343
```

(a) Desviaciones típicas en el tobillo derecho.

```
Posicion ángulo de la rodilla izquierda dentro del Ejercicio 1:  1.935345126888027
Posicion ángulo de la rodilla izquierda dentro del Ejercicio 2:  47.11223737097683
Posicion ángulo de la rodilla izquierda dentro del Ejercicio 3:  4.925706701561152
Posicion ángulo de la rodilla izquierda dentro del Ejercicio 4:  0.945936794050333
Posicion ángulo de la rodilla izquierda dentro del Ejercicio 5:  79.72272110616908
Posicion ángulo de la rodilla izquierda dentro del Ejercicio 6:  50.88405503360103
Posicion ángulo de la rodilla izquierda dentro del Ejercicio 7:  69.5738821169773
Posicion ángulo de la rodilla izquierda dentro del Ejercicio 8:  67.86025869366668
Posicion ángulo de la rodilla izquierda dentro del Ejercicio 9:  2.6232047144558504
Posicion ángulo de la rodilla izquierda dentro del Ejercicio 10:  1.3457824311694464
Posicion ángulo de la rodilla izquierda dentro del Ejercicio 11:  6.381063596330608
Posicion ángulo de la rodilla izquierda dentro del Ejercicio 12:  2.5420137291233695
Posicion ángulo de la rodilla izquierda dentro del Ejercicio 13:  1.132984618040629
Posicion ángulo de la rodilla izquierda dentro del Ejercicio 14:  3.0358892408942157
```

(b) Desviaciones típicas en el ángulo de la rodilla izquierda.

```
Posicion ángulo de la rodilla derecha dentro del Ejercicio 1:  1.259271331775088
Posicion ángulo de la rodilla derecha dentro del Ejercicio 2:  73.07131551197494
Posicion ángulo de la rodilla derecha dentro del Ejercicio 3:  10.396195375234816
Posicion ángulo de la rodilla derecha dentro del Ejercicio 4:  5.886210792823644
Posicion ángulo de la rodilla derecha dentro del Ejercicio 5:  87.05831837870869
Posicion ángulo de la rodilla derecha dentro del Ejercicio 6:  47.97195098897478
Posicion ángulo de la rodilla derecha dentro del Ejercicio 7:  104.46599388140754
Posicion ángulo de la rodilla derecha dentro del Ejercicio 8:  80.1325071121062
Posicion ángulo de la rodilla derecha dentro del Ejercicio 9:  3.1295132860041637
Posicion ángulo de la rodilla derecha dentro del Ejercicio 10:  1.1992037754498295
Posicion ángulo de la rodilla derecha dentro del Ejercicio 11:  18.274619543594735
Posicion ángulo de la rodilla derecha dentro del Ejercicio 12:  2.7115390384221287
Posicion ángulo de la rodilla derecha dentro del Ejercicio 13:  1.1058575364405814
Posicion ángulo de la rodilla derecha dentro del Ejercicio 14:  2.7118038287253574
```

(c) Desviaciones típicas en el ángulo de la rodilla derecha.

Figura 5.19: Pruebas sobre diferentes desviaciones típicas.

5.14. Clasificación de ejercicios

Una vez se han obtenido los valores correspondientes a las desviaciones típicas queda la parte de clasificarlas. Si se elige el ángulo de la rodilla derecha como punto de clasificación, habrá que agrupar los ejercicios que contengan una desviación típica elevada como ejercicios sobre las parte inferior del cuerpo, mientras que si cuentan con un valor bajo de desviación típica, se agruparán como ejercicios sobre la parte superior, pero ¿qué se considera un valor bajo o un valor alto?

Esta respuesta se puede responder con el concepto de deciles, cuartiles o percentiles.

1. **Cuartiles**, los cuartiles son tres valores $Q1$ $Q2$ $Q3$ que dividen el conjunto de datos y los ordenan en cuatro partes porcentualmente iguales.

2. **Deciles**, los deciles son nueve valores que dividen el conjunto de datos en diez partes porcentualmente iguales. El primer decil D_1 indica que solamente existe una probabilidad del 10 % de que la variable esté por debajo de esta cifra.

3. **Perentiles**, los percentiles son tal vez la medida más utilizada para el propósito de clasificación de características humanas. Los percentiles son 99 valores que dividen el conjunto en cien partes iguales de datos ordenados. El primer percentil P_1 indica que un valor supera el al uno por ciento de los valores y es superado por el noventa y nueve por ciento restante.

Para la clasificación de los ejercicios se ha elegido la medida de **percentil**. En la tabla 5.10 se puede observar un ejemplo de como el percentil clasifica con valores más altos los ejercicios que tienen una desviación típica mayor. De esta manera se podría lograr una clasificación de varios ejercicios.

Finalmente si lo que se espera es clasificar un único ejercicio la clasificación anterior nos es inválida ya que no se tienen otros ejercicios con los que comparar las desviaciones típicas. Lo que si que se obtiene son desviaciones típicas de distintas partes del cuerpo.

	desviación	DecileRank
Ejercicio1	17.254868	23
Ejercicio2	92.784492	84
Ejercicio3	25.000359	46
Ejercicio4	16.270868	15
Ejercicio5	124.640312	92
Ejercicio6	125.269585	99
Ejercicio7	88.113121	76
Ejercicio8	80.954239	69
Ejercicio9	12.105806	7
Ejercicio10	9.664043	0
Ejercicio11	17.749689	30
Ejercicio12	19.416697	38
Ejercicio13	27.311350	61
Ejercicio14	27.287348	53

Tabla 5.10: Clasificación de los ejercicios según el valor del **percentil**.

Para conseguir clasificar un ejercicio se pueden obtener las desviaciones típicas de partes superiores del cuerpo como las manos o los codos, y por otra parte, desviaciones típicas de las partes inferiores, como los tobillos o las rodillas. Si las desviaciones típicas de las partes superiores son mayores que las de las partes inferiores, quiere decir que se trata de un ejercicio sobre las partes superiores del cuerpo, de lo contrario sería sobre las partes inferiores. En la tabla 5.11 se puede observar un ejemplo de este tipo de clasificación y en la imagen 5.20 una serie de resultados de dichas clasificaciones.

	desviación rodilla izquierda	desviación rodilla derecha	desviación codo izquierdo	desviación codo derecho	Clasificación
Ejercicio1	1.935345	1.259271	83.475923	85.252242	Ejercicio SUPERIOR
Ejercicio2	47.112237	73.071316	16.182303	15.125709	Ejercicio INFERIOR
Ejercicio3	4.925707	10.396195	67.772181	71.215213	Ejercicio SUPERIOR
Ejercicio4	0.945937	5.886211	118.058889	116.983376	Ejercicio SUPERIOR
Ejercicio5	79.722721	87.058318	3.059813	2.034518	Ejercicio INFERIOR
Ejercicio6	50.884055	47.971951	3.406641	3.970406	Ejercicio INFERIOR
Ejercicio7	69.573882	104.465994	3.055756	1.953339	Ejercicio INFERIOR
Ejercicio8	67.860259	80.132507	2.067330	2.015728	Ejercicio INFERIOR
Ejercicio9	2.623205	3.129513	51.359515	51.526016	Ejercicio SUPERIOR
Ejercicio10	1.345782	1.199204	36.503302	36.661532	Ejercicio SUPERIOR
Ejercicio11	6.381064	18.274620	51.042877	54.722711	Ejercicio SUPERIOR
Ejercicio12	2.542014	2.711539	68.195441	68.669236	Ejercicio SUPERIOR
Ejercicio13	1.132985	1.105858	110.853035	116.354193	Ejercicio SUPERIOR
Ejercicio14	3.035889	2.711804	112.991661	118.296063	Ejercicio SUPERIOR

Tabla 5.11: Clasificación teórica de ejercicios.

Figura 5.20: Clasificación visual de ejercicios.

5.15. Aplicación de escritorio

Para concluir y poder comprobar visualmente los resultados de este proyecto se ha creado una aplicación de escritorio por medio de la interfaz de *Python* ***Tkinter***. Con el objetivo de poder mostrar un ejemplo más visual de las secuencias localizadas y como se podría adaptar el objetivo de este proyecto a un producto comercial.

En esta aplicación se podrán cargar tantos vídeos como el usuario desee. Habrá una sección dedicada a los vídeos concretos, es decir, a aquellos vídeos que realiza el terapeuta, otra sección con los vídeos completos, los que realizan los pacientes, y finalmente una sección en la que se mostrará el resultado final. Todos los vídeos se podrán reproducir y pausar en la misma aplicación.

Para ello se ha realizado una única pantalla en la que se pueden realizar las siguientes acciones:

1. Al iniciar la aplicación se cargarán los vídeos de las carpetas de referencia.
2. Los vídeos se reproducirán en la misma pantalla y cada uno en su sección correspondiente.
3. Las funciones de recorte dependerán del modo seleccionado. Si el usuario selecciona *modo ejemplo ON* los vídeos se recortarán automáticamente, mientras que si el usuario selecciona *modo ejemplo OFF* se deberá de especificar los *frames* de inicio y final del vídeo.
4. Todos los vídeos se podrán reproducir y pausar.
5. Una vez generado el recorte del vídeo, este mismo se guardará en la carpeta en la que se está ejecutando la aplicación.

Un ejemplo de de la aplicación en **modo ejemplo ON** es el que se muestra en la figura 5.22, mientras que un ejemplo de uso en **modo ejemplo OFF** correspondería con la figura 5.21. En ambas se puede observar como se selecciona un vídeo concreto, un vídeo completo que contiene varios ejercicios, y el programa devuelve el ejercicio concreto que se localiza dentro del vídeo que contiene múltiples ejercicios.

Para finalizar, especificar que aunque esta parte del proyecto sea la más vistosa no es para nada la parte más relevante del proyecto ya que ha sido creada únicamente para mostrar los resultados. Esta aplicación

Figura 5.21: Recorte de vídeos en modo ejemplo OFF.

Figura 5.22: Recorte de vídeos en modo ejemplo ON.

podría ser usada por un terapeuta que ha obtenido múltiples secuencias de inicio y fin y quiere comprobar como de bien están realizando sus pacientes los ejercicios. Pero esta cuestión queda únicamente planteada ya que de momento el algoritmo no es lo suficientemente preciso como para que se comercialice.

5.16. Aspecto relevante

Finalmente comentar en este apartado que todas las imágenes en las que aparecen individuos han sido obtenidas por medio de la web **Pixabay** que ofrece multitud de imágenes con gran calidad y sin derechos de autor, o han sido creadas por la autora del proyecto tras recibir el consentimiento de sus compañeros para mostrar sus vídeos e imágenes en este proyecto.

Trabajos relacionados

El progreso de la calidad de vida de pacientes con la enfermedad de *Parkinson* mejora con el paso del tiempo y eso es debido a las múltiples investigaciones que se hacen sobre ello. Un ámbito en el que es fundamental avanzar es en la creación de sistemas de rehabilitación *online*.

La creación de sistemas que por medio de la visión artificial puedan evaluar ejercicios de un paciente, hace que tanto los pacientes como el terapeuta reciban una retroalimentación inmediata y precisa de los ejercicios que se realizan.

A continuación se van a exponer algunas de las investigaciones que van aportando nuevos conocimientos en la mejora de las técnicas existentes.

Feasibility of Using Dynamic Time Warping to Measure Motor States in Parkinson's Disease [2]

En este artículo publicado por las universidades de Dalarna y Halmstad, ambas en Suiza, estudia la viabilidad de aplicar **DTW** para el análisis de los estados motores en pacientes con la enfermedad de *Parkinson*. Para ello se implantó un dispositivo en el tobillo de cada uno de los 19 pacientes que participaron en el proceso.

Este estudio persigue evaluar los síntomas motores de los pacientes, y como los mismos suelen sufrir caídas, está sobretodo enfocado a detectar los movimientos de las extremidades inferiores del cuerpo humano. Los datos fueron extraídos de diferentes señales de golpeo del pie que ejercían los pacientes sobre el suelo, en concreto se almacenaban seis señales por cada golpeo.

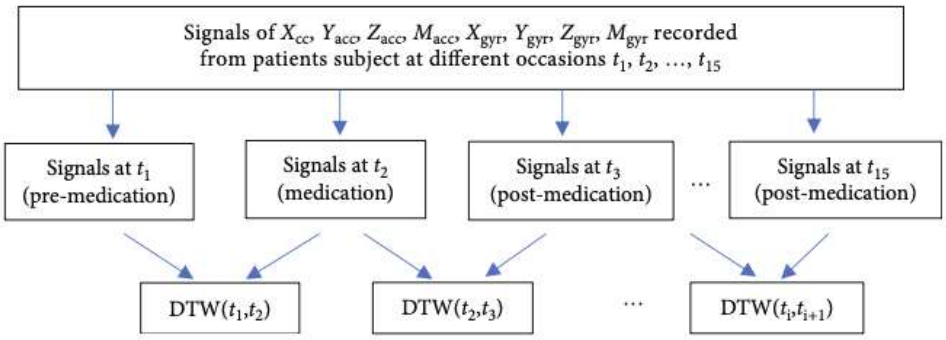


Figura 6.1: Arquitectura del conjunto de datos.

El conjunto de datos fue creado a partir de las seis señales, X_{acc}, Y_{acc}, Z_{acc} representan la aceleración de golpeo del pie, y X_{gyr}, Y_{gyr}, Z_{gyr} para representar el giroscopio. Las medidas de distancia fueron calculadas cada dos pruebas consecutivas, como se puede observar en la figura 6.1 y se construyó una puntuación de distancia del estado motor, DDS.

Una serie de especialistas en trastornos del movimiento clasificaron los estados motores de los pacientes de acuerdo a la TRS, *Escala de Respuesta al Tratamiento*. Con los datos generados por pacientes que sufrían la enfermedad de *Parkinson* se pudo concluir que el DDS medio fue capaz de identificar entre pacientes con EP en estado avanzado y clasificar los cambios del estado motor con buena precisión.

Por último se identificaron características basadas en DTW para cada paciente, por lo que se puede concluir que por medio de esta disciplina se puede proporcionar información sobre el estado motor de los pacientes con EP avanzada.

A Dynamic Time Warping Based Algorithm to Evaluate Kinect-Enabled Home-Based Physical Rehabilitation Exercises for Older People [38]

En este artículo redactado por el Departamento de Ingeniería Industrial y de Sistemas del Instituto Avanzado de Ciencia y Tecnología de Corea, se puede observar el uso de técnicas de visión por computador para calcular características sobre las posicione humanas en distintos instantes de tiempo.

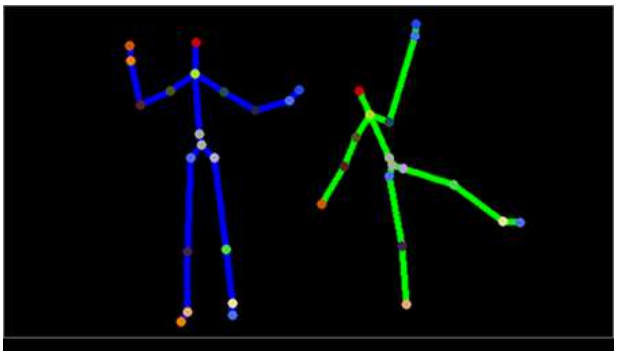


Figura 6.2: Obtención del esqueleto mediante la tecnología *Kinect*.

A través de la tecnología ***Kinect*** [5] el movimiento humano es almacenado en forma de coordenadas. Estas coordenadas serán analizadas para determinar una similitud entre el movimiento del entrenador y el movimiento del usuario final. En la figura 6.2 se puede observar como se representarían los esqueletos obtenidos mediante esta técnica

De los esqueletos anteriores, para la comparación de posiciones solo se usan las relativas a las extremidades superiores e inferiores. El primer paso que toma el algoritmo es cuantificar la diferencia entre cada una de estas poses del entrenador con las del usuario final. En la figura 6.3 se puede comprender como a través de la ley del coseno se puede calcular la diferencia entre la posición del entrenador y del aprendiz.

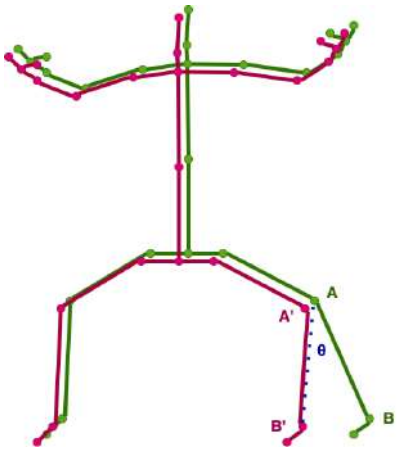


Figura 6.3: Diferencia de posiciones en esqueletos.

[5] *Kinect* es un sistema desarrollado por Microsoft que permite reconocer la figura de un individuo en 3D a partir de una cámara RGBD

La comparación de movimientos se realiza, una vez localizadas las dos secuencias a comparar, se aplica DTW para determinar la coincidencia óptima. Se obtendrían dos matrices de ocho filas y n columnas, siendo n el número de movimientos realizados por el individuo y ocho las posiciones óseas correspondientes a las extremidades superiores e inferiores. Además se tuvo en cuenta la orientación del individuo.

Los resultados obtenidos en este estudio son una puntuación de rendimiento adquirida a través del algoritmo DTW en términos de porcentaje. Sobre unos ejercicios de *Tai Chi* en un total de 21 participantes, se llegó a la conclusión de que el algoritmo basado en DTW podría utilizarse eficazmente para la evaluación autónoma del rendimiento en diferentes ejercicios de rehabilitación en el hogar.

Conclusiones y Líneas de trabajo futuras

Para concluir con la exposición de esta memoria, se van a comentar tanto las conclusiones del trabajo como las posibles mejoras o líneas futuras que se podrían incluir en un futuro.

7.1. Conclusiones

De este proyecto se han obtenido varias conclusiones:

Como primera conclusión cabe destacar que continuar con un trabajo de otros compañeros puede resultar fácil pero adaptar un proyecto del que desconoces su funcionamiento y sobretodo la tecnología utilizada ha resultado en ocasiones bastante complicado.

Por otra parte, el problema principal de esta proyecto ha sido el tiempo necesario para obtener los resultados. Por una parte el tiempo empleado en extraer las secuencias, ya que este tiempo no depende únicamente de las ejecuciones del programa si no que muchas veces tras varios intentos se veía que algo no se estaba implementando correctamente y había que restaurar las imágenes *docker*, aspecto que consumía una demora en la obtención de los resultados. Y por otra parte el tiempo empleado en obtener el resultado. En muchas ocasiones es difícil realizar las pruebas por el tiempo que requieren y hay que tener en cuenta que para llegar a la solución se ha necesitado implementar múltiples algoritmos, probar con datos de gran tamaño, etc. Combinar todo esto ha creado situaciones en las que la búsqueda de la solución ha sido muy tediosa y desesperante, ya que muchas veces

las ejecuciones eran demasiado largas y para concluir los resultado nada esperanzadores.

A pesar de todos los inconvenientes me ha encantado el proyecto. Una vez vas solucionando los problemas y consigues resultados favorables, te invitan a seguir investigando y pensando nuevas formas de hacerlo más eficiente o preciso.

7.2. Líneas futuras

En este proyecto se han realizado múltiples pruebas para poder encontrar un patrón de referencia dentro de una secuencia de mayor tamaño. También se ha conseguido detectar si el ejercicio en cuestión corresponde a las extremidades superiores o inferiores. Con estas aportaciones se pueden proponer las siguientes líneas futuras:

1. Una vez reconocido el ejercicio se podría evaluar como de bien o mal está realizado. En un principio se planteó utilizar la distancia o el coste que proporciona el algoritmo que encuentra el camino óptimo para devolver como de parecida es la secuencia encontrada en comparación con la de referencia. Los resultados obtenidos fueron desalentadores por lo que la evaluación de ejercicios tendría que realizarse de otra manera.

2. Clasificar el tipo de ejercicios en un rango más amplio, no solo si se trata de extremidades superiores o inferiores.

3. Esperar a que evolucionen los algoritmos de detección de objetos y poder hacer pruebas con esqueletos que hayan sido obtenidos de una manera más precisa.

4. Calcular la diferencia entre movimientos. Un aspecto que puede ser interesante a la hora de encontrar la secuencia es mediante el cálculo de la diferencia de posición. Si sobre el patrón de referencia almacenamos los valores relativos a los cambios de posición de cada una de las extremidades entre el *frame* actual y el inmediatamente anterior, se podría obtener una secuencia muy reducida y precisa con la que posteriormente realizar los cálculos de búsqueda.

5. Ejecutar la búsqueda de secuencias una vez sea clasificado el ejercicio según la extremidad del cuerpo que se ejercite en el mismo. De esta

forma se le podrá dar mayor peso a la parte del cuerpo ejercitada y después implementar la búsqueda de secuencias.

6. Aplicar el algoritmo obtenido a vídeos reales, realizados por terapeutas profesionales y pacientes con la enfermedad de *Parkinson*.

7. La aplicación diseñada es una mera muestra de datos para presentar de una forma más visual el fin de este proyecto al jurado pero esta misma aplicación podría ser mejorada y distribuida a los terapeutas para que evalúen ellos mismos como están realizando los ejercicios sus pacientes.

8. Automatizar la aplicación de escritorio para que saque las secuencias sin la necesidad de especificarle los *frames*.

Bibliografía

[1] Ensemble of convolutional neural networks based on an evolutionary algorithm applied to an industrial welding process. *Computers in Industry*, 133:103530, 2021.

[2] Somayeh Aghanavesi, Hasan Fleyeh, and Mark Dougherty. Feasibility of using dynamic time warping to measure motor states in parkinson's disease. *Journal of Sensors*, 2020, 2020.

[3] V Arévalo, J González, and G Ambrosio. La librería de visión artificial opencv. aplicación a la docencia e investigación. *Base Informática*, 40:61–66, 2004.

[4] Douglas Beniz, Alexey Espindola, et al. Using tkinter of python to create graphical user interface (gui) for scripts in lnls. *WEPOPRPO25*, 9:25–28, 2016.

[5] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[6] Carmelo Bonilla Carrión. Redes convolucionales. 2020.

[7] Qi Dang, Jianqin Yin, Bin Wang, and Wenqing Zheng. Deep learning based 2d human pose estimation: A survey. *Tsinghua Science and Technology*, 24(6):663–676, 2019.

[8] Bethesda (MD): Biblioteca Nacional de Medicina (EE. UU.). Enfermedad de parkinson, [actualizado 1 mayo 2019; revisado 30 oct. 2018; consulta 30 ago. 2019].

[9] Instituto Nacional de Trastornos Neurológicos y accidentes Cerebrovasculares. Enfermedad de parkinson. https://espanol.ninds.nih.gov/es/trastornos/enfermedad-de-parkinson, 2022.

[10] Docker docs. Docker overview. https://docs.docker.com/get-started/overview/, 2022.

[11] Kaiming He Georgia Gkioxari Piotr Dollár and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[12] Parra Medina Luis Enrique, Arankowsky Sandoval Gloria, Salazar Ceballos Jorge Efraín, and Góngora Alfaro José Luis. Latencia diagnóstica en la enfermedad de parkinson y su relación con los síntomas prodrómicos motores y no motores. *eNeurobiología*, 10(25):1, 2019.

[13] Usama M Fayyad, David Haussler, and Paul E Stolorz. Kdd for science data analysis: Issues and examples. In *KDD*, pages 50–56, 1996.

[14] Python Software Foundation. Usando pypi, time series distances. https://pypi.org/project/pydtw, Apr 26, 2019.

[15] Python Software Foundation. Usando pypi, pydtw. https://pypi.org/project/dtaidistance, Jun 20,2022.

[16] Iván García and Víctor Caranqui. La visión artificial y los campos de aplicación. *Tierra infinita*, 1(1):98–108, 2015.

[17] R García-Ramos, E López Valdés, L Ballesteros, S de Jesús, and P Mir. Informe de la fundación del cerebro sobre el impacto social de la enfermedad de parkinson en españa. *Neurología*, 31(6):401–413, 2016.

[18] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. https://github.com/facebookresearch/detectron, 2018.

[19] Wenjuan Gong, Xuena Zhang, Jordi Gonzàlez, Andrews Sobral, Thierry Bouwmans, Changhe Tu, and El-hadi Zahzah. Human pose estimation from monocular images: A comprehensive survey. *Sensors*, 16(12), 2016.

[20] Lauren Hirsch, Nathalie Jette, Alexandra Frolkis, Thomas Steeves, and Tamara Pringsheim. The incidence of parkinson's disease: a systematic review and meta-analysis. *Neuroepidemiology*, 46(4):292–300, 2016.

[21] Brijnesh J Jain and David Schultz. Optimal warping paths are unique for almost every pair of time series. *arXiv preprint arXiv:1705.05681*, 2017.

[22] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[23] Ana González Marcos, Francisco Javier Martínez de Pisón Ascacíbar, Fernando Alba Elías, Manuel Castejón Limas, Joaquín Bienvenido Ordieres Meré, Eliseo Pablo Vergara González, et al. Técnicas y algoritmos básicos de visión artificial. *Técnicas y Algoritmos Básicos de Visión Artificial*, 2006.

[24] José Alberto Mauricio. Análisis de series temporales. *Universidad Complutence de Madrid*, 2007.

[25] Wes McKinney et al. pandas: a foundational python library for data analysis and statistics. *Python for high performance and scientific computing*, 14(9):1–9, 2011.

[26] Alex Moroz, Steven R. Edgley, Henry L. Lew, John Chae, Lisa A. Lombard, Cara Camiolo Reddy, and Keith M. Robinson. Rehabilitation interventions in parkinson disease. *PMR*, 1(3, Supplement):S42–S48, 2009.

[27] Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

[28] Gabriel Ortiz-Genaro, Héctor González-Usigli, Fermín Paul Pacheco-Moisés, Irma Ernestina Velázquez-Brizuela, Angélica Lizeth Sánchez-López, Erika Daniela González-Renovato, Aidé Irene Mesa-Acuña, Blanca Miriam Torres-Mendoza, and Daniela Lucero Delgado-Lara. Diferencias de género en pacientes con depresión y ansiedad con enfermedad de parkinson. *Archivos de Neurociencias*, 25(1):51–60, 2020.

[29] César Pérez López and Daniel Santin González. *Minería de datos. Técnicas y herramientas: técnicas y herramientas.* Editorial Paraninfo, 2007.

[30] Anumeet Priyadarshi, Sadik A Khuder, Eric A Schaub, and Snigdha S Priyadarshi. Environmental risk factors and parkinson's disease: a metaanalysis. *Environmental research*, 86(2):122–127, 2001.

[31] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[32] Roberta Arb Saba, Débora Palma Maia, Francisco Eduardo Costa Cardoso, Vanderci Borges, Luiz Augusto F Andrade, Henrique Ballalai Ferraz, Egberto Reis Barbosa, Carlos Roberto de Mello Rieder, Delson José da Silva, Hsin Fen Chien, et al. Guidelines for parkinson's disease treatment: consensus from the movement disorders scientific department of the brazilian academy of neurology-motor symptoms. *Arquivos de Neuro-Psiquiatria*, 80:316–329, 2022.

[33] Romain Tavenard, Johann Faouzi, Gilles Vandewiele, Felix Divo, Guillaume Androz, Chester Holtz, Marie Payne, Roman Yurchak, Marc Rußwurm, Kushal Kolar, et al. Tslearn, a machine learning toolkit for time series data. *J. Mach. Learn. Res.*, 21(118):1–6, 2020.

[34] Mrinal Walia. Una guía completa sobre la estimación de la pose humana, [actualizado 115 mayo 2019; revisado 10 nov. 2018; consulta 10 abr. 2022].

[35] Wikipedia. Latex — wikipedia, la enciclopedia libre, 2015. [Internet; descargado 30-septiembre-2015].

[36] Karin Wirdefeldt, Hans-Olov Adami, Philip Cole, Dimitrios Trichopoulos, and Jack Mandel. Epidemiology and etiology of parkinson's disease: a review of the evidence. *European journal of epidemiology*, 26(1):1–58, 2011.

[37] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

[38] Xiaoqun Yu and Shuping Xiong. A dynamic time warping based algorithm to evaluate kinect-enabled home-based physical rehabilitation exercises for older people. *Sensors*, 19(13), 2019.

[39] Jeremy Zhang. Dynamic time warping. *Towards Data Science. https://towardsdatascience. com/dynamic-time-warping-3933f25fcdd. Retrieved Oct*, 8:2021, 2020.