

Universidades de Burgos, León y  
Valladolid

Máster universitario

## **Inteligencia de Negocio y Big Data en Entornos Seguros**



**Trabajo Fin de Máster**

**Detección y comparación de  
posturas en un flujo de imágenes**

Presentado por José Miguel Ramírez Sanz  
en Universidad de Burgos — 28 de junio  
de 2020

Tutor: Dr. José Francisco Díez Pastor  
Dr. Álvar Arnaiz González





# Universidades de Burgos, León y Valladolid



## Máster universitario en Inteligencia de Negocio y Big Data en Entornos Seguros

D. José Francisco Díez Pastor, profesor del departamento de Ingeniería Informática, área de Lenguajes y Sistemas Informáticos.

Expone:

Que el alumno D. José Miguel Ramírez Sanz, con DNI 71303106R, ha realizado el Trabajo final de Máster en Inteligencia de Negocio y Big Data en Entornos Seguros titulado TFM.

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, 28 de junio de 2020

Vº. Bº. del Tutor:

Dr. José Francisco Díez Pastor

Vº. Bº. del co-tutor:

Dr. Álvar Arnaiz González





## Resumen

El **Parkinson** es un trastorno neurodegenerativo que afecta al sistema nervioso de la persona. Esta enfermedad afecta principalmente a **personas mayores**. Una de las mejores formas de mejorar la vida de los pacientes que padecen esta enfermedad son las **rehabilitaciones**, donde un médico o terapeuta ayuda a mejorar la **movilidad** y la **autonomía** del paciente.

En la actualidad, estas rehabilitaciones se hacen de forma **presencial**, teniendo que ser el paciente el que se **desplace** hasta el **centro de salud**. Además, con la constante **despoblación** que sufren las **zonas rurales** de España, el número de centro de salud se ha reducido considerablemente, haciendo así que los pacientes tengan que **desplazarse** a otras localidades para hacer sus rehabilitaciones, que teniendo en cuenta su dependencia y su elevada edad es un **gran problema**.

Es por ello que surgió el **proyecto** «Estudio de factibilidad y coste-efectividad del uso telemedicina con un equipo multidisciplinar para enfermedad de Parkinson», donde se ha desarrollado una aplicación que permite realizar rehabilitaciones de manera **online** sin necesidad de que el paciente se desplace a un centro de salud.

Dentro del desarrollo de este proyecto se encuentra el **subproyecto** que se ha realizado en este Trabajo Fin de Máster, un **estudio del estado del arte** para la selección de un modelo de **visión artificial** capaz de estimar la posición de una persona en una imagen, y posterior **implementación** de un **sistema de extracción de características y comparación de posiciones**. Todo ello con la **finalidad** de poder comparar los ejercicios realizados por el paciente con los grabados por los terapeutas. Este desarrollo permite la **automatización** de las rehabilitaciones **online**, así como una continua **retroalimentación** de la evolución del paciente **mejorando** así su **frecuencia** de trabajo y su **motivación** por mejorar.

## Descriptores

*Parkinson, rehabilitación, flujo de datos, Big Data, visión artificial, investigación.*

## Abstract

**Parkinson** is a neurodegenerative disorder that affects the nervous system of the person. This illness affects mainly to **old people**. One of the best ways to improve patient's lives that suffer this illness is **rehabilitation**, where a doctor or a therapist helps to improve the patient's **mobility** and **autonomy**.

Nowadays, rehabilitations are made **face-to-face**, been the patient the one who has to **travel** to the **clinic**. Furthermore, with the constant **depopulation** that suffers **rural zones** of Spain, the number of clinics has been reduced considerably, causing patients to have to **move** to others locations to do their rehabilitations, that considering their dependency and their old age, it is a **big problem**.

That is why “Feasibility and cost-effectiveness study of telemedicine use with a multidisciplinary team for Parkinson’s disease” **project** surged, where an application has been developed to allow **online** rehabilitation without the need for the patient to move to the clinic.

Inside this project, there is a **subproject** that has been done in this End of Master’s Project, a **state of art study** to select the best **computer vision** model to estimate the posture of a person in an image, and the **implementation** of an **feature extraction system and postures comparation**. All with the **purpose** of to be able to compare patient’s exercises to therapist’s exercises. This development allows the **automation** of online rehabilitations, as well as continuous **feedback** on the patient’s evolution, thus **improving** their exercise **frequency** and their **motivation** to improve.

## Keywords

*Parkinson, rehabilitation, data stream, Big Data, computer vision, research.*

---

# Índice general

---

Índice general	III
Índice de figuras	VI
Índice de tablas	VIII
<b>Memoria</b>	<b>1</b>
<b>1. Introducción</b>	<b>3</b>
1.1. Estructura de la memoria . . . . .	5
1.2. Estructura de los apéndices . . . . .	6
<b>2. Objetivos del proyecto</b>	<b>7</b>
2.1. Objetivos funcionales . . . . .	7
2.2. Objetivos técnicos . . . . .	7
2.3. Objetivos personales . . . . .	8
<b>3. Conceptos teóricos</b>	<b>9</b>
3.1. Vídeo . . . . .	9
3.2. Visión por computador . . . . .	10
3.3. Redes neuronales . . . . .	11
3.4. Puntos y vectores . . . . .	20
<b>4. Técnicas y herramientas</b>	<b>23</b>
4.1. Visión por computador ~ Detección de movimiento . . . . .	23
4.2. Estación de trabajo . . . . .	24

<b>5. Aspectos relevantes del desarrollo del proyecto</b>	<b>25</b>
5.1. Desarrollo FIS-HUBU . . . . .	25
5.2. Investigación algoritmos de visión por computador . . . . .	29
5.3. Investigación de <i>Detectron2</i> . . . . .	30
5.4. Cálculo de características . . . . .	36
5.5. Primeras versiones comparación de posiciones . . . . .	38
5.6. Versión 3 de comparación, uso de medias . . . . .	42
5.7. Error en brazos, nueva posición y comparación . . . . .	42
5.8. Comprobación de la última versión . . . . .	48
5.9. Versión reducida . . . . .	51
5.10. Preparación para flujos . . . . .	52
5.11. Implementación en el flujo . . . . .	52
5.12. Estudio final del flujo . . . . .	54
<b>6. Trabajos relacionados</b>	<b>61</b>
<b>7. Conclusiones y Líneas de trabajo futuras</b>	<b>65</b>
7.1. Conclusiones . . . . .	65
7.2. Líneas de trabajo futuras . . . . .	66
<b>Apéndices</b>	<b>68</b>
<b>Apéndice A Plan de Proyecto Software</b>	<b>71</b>
A.1. Introducción . . . . .	71
A.2. Planificación temporal . . . . .	71
A.3. Estudio de viabilidad . . . . .	75
<b>Apéndice B Especificación de diseño</b>	<b>77</b>
B.1. Introducción . . . . .	77
B.2. Diseño de datos . . . . .	77
B.3. Diseño procedimental . . . . .	78
<b>Apéndice C Documentación técnica de programación</b>	<b>81</b>
C.1. Introducción . . . . .	81
C.2. Estructura de directorios . . . . .	82
C.3. Manual del programador . . . . .	84
C.4. Compilación, instalación y ejecución del proyecto . . . . .	84
C.5. Pruebas del sistema . . . . .	87
<b>Apéndice D Documentación de usuario</b>	<b>89</b>
D.1. Introducción . . . . .	89

## *ÍNDICE GENERAL*

v

D.2. Requisitos de usuarios . . . . .	89
D.3. Instalación . . . . .	89
D.4. Manual del usuario . . . . .	89
<b>Bibliografía</b>	<b>91</b>

---

# Índice de figuras

---

3.1. Ejemplos tipos de utilidades de la visión por computador. . . . .	11
3.2. Organización de una red neuronal convolucional [19]. . . . .	12
3.3. Ejemplo aplicación de un filtro con <i>stride</i> 1 [19]. . . . .	14
3.4. <i>Dilated Convolution</i> con dilatación a 2 [19]. . . . .	16
3.5. Ejemplo de capa de agrupación de tamaño 2 [19]. . . . .	17
3.6. Funciones de activación más comunes en capas de no linealidad [19].	18
3.7. Ejemplo bloque residual en ResNet [19]. . . . .	19
3.8. Region-Based CNN [19]. . . . .	19
3.9. Feature Pyramid Network [21]. . . . .	20
5.10. Menú principal de un responsable. . . . .	27
5.11. Menú de estadísticas. . . . .	27
5.12. Ejemplo de la evolución de un paciente vista por un responsable.	28
5.13. Menú principal de los pacientes. . . . .	28
5.14. Mejora del rendimiento en GB/s con NVlink [24]. . . . .	31
5.15. Modelos <i>Detectron2</i> . . . . .	32
5.16. Modelos <i>Detectron2</i> . . . . .	33
5.17. Modelos <i>Detectron2</i> . . . . .	33
5.18. Pruebas con ropa y objetos. . . . .	35
5.19. Pruebas con el <i>threshold</i> . . . . .	35
5.20. Puntos clave predichos con el modelo junto con la etiqueta con su orden. . . . .	37
5.21. Ejemplo ángulos sobre los que se calcula la diferencia entre posiciones. . . . .	40
5.22. Ejemplos problemas con codos. . . . .	40
5.23. Ejemplos problemas con hombros. . . . .	41
5.24. Diagrama de flujos penalización comparación versión 2. . . . .	41
5.25. Ejemplo de error en la comparación de brazos. . . . .	43

5.26. Resultado ejecución de solo la posición y comparación del flujo en milisegundos. . . . .	56
5.27. Distribución tiempo medio de ejecución solo posición y comparación. . . . .	57
5.28. Resultado ejecución flujo en milisegundos. . . . .	58
5.29. Distribución tiempo medio de ejecución flujo. . . . .	59
6.30. Ejemplos cámaras en rehabilitación de manos [33]. . . . .	63
6.31. Disposición <i>Kinects V2(K)</i> , <i>NUC(N)</i> y el <i>Master PC</i> [3]. . . . .	64
B.1. Diagrama de clases. . . . .	78
B.2. Diagrama de secuencia, estimación posición. . . . .	79
B.3. Diagrama de secuencia, comparación de posiciones. . . . .	80

---

# Índice de tablas

---

5.1.	Tabla con el estudio de los modelos de posición ordenado por ratio.	34
5.2.	Tabla con las partes de la posición y su identificador. . . . .	37
5.3.	Tabla con las estadísticas de las diferencias en grados. . . . .	49
5.4.	Tabla con las estadísticas de las diferencias en porcentajes de exactitud. . . . .	49
5.5.	Tabla con los tiempos de la ejecución. . . . .	49
5.6.	Tabla con los tiempos de la ejecución con la versión reducida de la posición. . . . .	52
5.7.	Tabla con los resultados de la ejecución solo de la posición y comparación del flujo en milisegundos. . . . .	55
5.8.	Tabla con los resultados de la ejecución del flujo en milisegundos.	57
A.1.	Tabla con las licencias de las librerías y herramientas utilizadas.	76

# **Memoria**



---

# Introducción

---

La enfermedad del **Parkinson** es trastorno neurodegenerativo (enfermedades que afectan al cerebro humano, con carácter **crónico**, es decir no se puede curar, y **progresivo**, es decir, que la enfermedad evoluciona [27, 8]) que afecta al **sistema nervioso** de la persona. Pertenece al grupo de enfermedades llamadas **Trastornos del Movimiento**, como la enfermedad de *Huntington* o el síndrome de las piernas inquietas [6, 13, 5, 23].

La enfermedad del *Parkinson* tiene diversos síntomas con los cuales se puede detectar, entre estos están los **temblores**, que es el síntoma más característico de la enfermedad, pérdida de los movimientos automáticos, rigidez muscular y lentitud en los movimientos, problemas de equilibrio y/o coordinación...

Las principales causas de la enfermedad de *Parkinson*, que recibe su nombre por el doctor que la descubrió *James Parkinson* en 1817, son:

- **Factor genético.** Se estima que entre un 15 % y 25 % de las personas que sufren la enfermedad tienen algún pariente que ha sufrido o está sufriendo la enfermedad.
- **Factores ambientales.** Estar expuesto a alguna toxina como puede ser un pesticida.

Además, existen una serie de factores que aumentan la probabilidad de tener la enfermedad, estos son:

- **Edad.** Las personas mayores de 60 años tienen una probabilidad de sufrir la enfermedad.

- **Sexo.** Los hombres son más propensos a sufrir la enfermedad.

Los pacientes pueden ser clasificados en 5 tipos distintos llamados **estadios**. En el estadio 1 el paciente solo sufre una afectación en un lateral de su cuerpo, mientras que en el último estadio el paciente tiene una gran afectación en todo el cuerpo además de no poder ser una persona autosuficiente.

El *Parkinson* es la segunda enfermedad neurodegenerativa más **común** en el mundo, después del *Alzhéimer*. En total hay **6,5 millones** de personas afectas por *Parkinson* en el mundo, de las cuales entre 160 mill y 300 mil están en España [10].

Para poder tratar a los pacientes de esta enfermedad existen cuatro tipos de **tratamientos** que pueden ser complementarios, estás son farmacológicos, quirúrgico, terapias avanzadas y no farmacológicos. Es en este último donde se encuentra uno de los tratamientos más sencillos y que mayor beneficio aporta a los pacientes, las **rehabilitaciones**.

Estas **rehabilitaciones** se realizan mayoritariamente en las consultas médicas, teniendo que desplazarse el paciente (que habitualmente es una persona muy mayor con grandes dificultades para moverse, sobre todo en los estadios más avanzados) hasta allí. Además, con la creciente **despoblación** de las **zonas rurales** los centros de salud de los municipios más pequeños se están cerrando por falta de recursos y pacientes a los que atender. Cabe destacar también la influencia de pandemias como la que se está viviendo actualmente con el *COVID-19*, que al no poder dirigirse los pacientes a los centros de salud por la saturación del sistema sanitario y al ser personas de riesgo, las rehabilitaciones de los pacientes se posponen de forma indefinida.

Es por ello que el avance en las **tecnologías** de telecomunicación han permitido el desarrollo de **rehabilitaciones online**, en las cuales el paciente no ha de movilizarse a ningún sitio, sino que puede realizar sus rehabilitaciones correspondientes desde su casa. Además, los constantes avances que se están dando en área de la informática como el **Big Data** y la **visión por computador** están permitiendo desarrollar los primeros prototipos de rehabilitaciones *online* sin necesidad de la presencia de un terapeuta o un médico.

Es de aquí donde nació el proyecto «Estudio de factibilidad y coste-efectividad del uso telemedicina con un equipo multidisciplinar para enfermedad de Parkinson», un proyecto de colaboración entre el Hospital Universitario de Burgos y la Universidad de Burgos, para desarrollar un sistema informático que permite realizar rehabilitaciones *online* entre paciente

y terapeuta, y que además permita la realización de ejercicios de forma autónoma por parte del paciente mostrándole su valoración y evolución de sus ejercicios. Todo ello bajo una interfaz y una interacción con los dispositivos **sencilla** para los pacientes.

El desarrollo de la aplicación web para el soporte del sistema fue desarrollado en conjunto con José Luis Garrido Labrador, como se comenta en el apartado 5.1. El resto del desarrollo se dividió en:

- Desarrollo de un **flujo extensible** capaz de recoger la imagen gravada en el dispositivo del paciente y una vez en el servidor aplicar métodos para la mejora y la anonimización de la imagen. Este punto ha sido desarrollado por el compañero José Luis Garrido Labrador.
- Investigación de una herramienta capaz de **estimar la postura** del paciente e implementar un sistema de **interpretación y comparación** de posiciones capaz de valorar la similitud entre ejercicios. Todo ello debía de poder ser ejecutado en el flujo del compañero de forma que se tardará el menos tiempo posible en la ejecución de todas las partes y con el menor uso de memoria posible.

Es sobre este último punto el que se desarrolla en este documento, la implementación de un sistema de **estimación y comparación** de posiciones que trabaje con una **gran cantidad de datos, muy pesados** y que pueda ejecutarse en un **flujo**.

## 1.1. Estructura de la memoria

La presente memoria se compone de los siguientes apartados:

1. **Introducción:** descripción de la enfermedad de *Parkinson*, introducción al proyecto y al trabajo desarrollado.
2. **Objetivos del proyecto:** objetivos funcionales, técnicos y personales que se fijaron al comienzo del proyecto.
3. **Conceptos teóricos:** definiciones y explicaciones de las distintas nociones teóricas necesarias para la correcta comprensión del trabajo realizado.
4. **Técnicas y herramientas:** conjunto de herramientas utilizadas en el desarrollo del proyecto.

5. **Aspectos relevantes del desarrollo:** documentación de los sucesos más importantes en el desarrollo del proyecto.
6. **Trabajos relacionados:** conjunto de proyectos similares en los que se fijó en las distintas partes del desarrollo.
7. **Conclusiones y Líneas de trabajo futuras:** conclusiones finales del proyecto y las posibles mejoras que se podrían haber realizado si se tuviese más tiempo o más recursos económicos.

## 1.2. Estructura de los apéndices

Los apéndices del documento se organizan de la siguiente manera:

1. TODO.

---

# **Objetivos del proyecto**

---

En este apartado se van a detallar los objetivos funcionales, técnicos y personales que se tienen en el desarrollo de este proyecto.

## **2.1. Objetivos funcionales**

Los distintos objetivos que se querían cumplir con el desarrollo del proyecto fueron:

- Desarrollar una aplicación web para poder realizar y evaluar las rehabilitaciones *online* de los pacientes con *Parkinson*.
- Desarrollar una aplicación web accesible y fácil de utilizar, sobre todo para los pacientes.
- Realizar un estudio del estado del arte de las herramientas de visión artificial y estimación de posición de personas.
- Crear una herramienta capaz de comparar las rehabilitaciones hechas por los pacientes con los ejercicios bases.

## **2.2. Objetivos técnicos**

En este subapartado se va a detallar los distintos objetivos relacionados con las técnicas y herramientas que se quieren aprender y utilizar:

- Crear una aplicación web para realizar rehabilitaciones *online*.

- Crear una herramienta de comparación de ejercicios a partir de algoritmos de visión por computador y detección de movimientos, que permita comparar los fotogramas de un mismo vídeo.
- Desarrollar una herramienta de comparación que utilice el mínimo tiempo y memoria posible para poder trabajar con flujos de datos.
- Utilizar el lenguaje de programación *Python* para la comparación de ejercicios, donde se incluyen tareas de inteligencia artificial para la detección de movimiento, extracción de características...
- Realizar el desarrollo del proyecto utilizando un repositorio *Git*, en concreto en *GitHub* para poder controlar las tareas y las versiones del proyecto.
- Desplegar el sistema en dispositivos con un valor ajustado.
- Utilizar la extensión de *Git* llamada *ZenHub* para controlar el estado de las tareas y la temporalidad de estas, y así seguir el modelo *SCRUM* para desarrollar el proyecto de forma incremental.

### **2.3. Objetivos personales**

Por último, se van a comentar los objetivos personales que se tienen en este proyecto, donde se encuentran objetivos desde probar conocimientos adquiridos en el máster como mejorar algunas cualidades personales.

- Poder ayudar a las personas mayores con Parkinson para que puedan realizar las rehabilitaciones necesarias sin necesidad de salir de sus casas. Además, mejorar esta rehabilitación a partir de la comparación de los ejercicios realizados.
- Mejorar mis capacidades comunicativas y de exposición en las diversas presentaciones del proyecto a los responsables del hospital universitario, como a los pacientes a los que se instale los distintos dispositivos del proyecto.
- Usar los conocimientos adquiridos durante la carrera y durante el máster.
- Mejorar mis conocimientos sobre visión por computador.
- Conocer los distintos algoritmos de comparación y clasificación de instancias para los datos obtenidos sobre los ejercicios.

---

# **Conceptos teóricos**

---

En este apartado se van a explicar los distintos conceptos teóricos necesarios para comprender correctamente el proyecto.

## **3.1. Vídeo**

El vídeo es un tipo de datos compuesto por un conjunto de imágenes, y en algunos casos, una señal de audio sincronizada con las imágenes. Existen diversos formatos para vídeos que dependen de la codificación de los datos que utiliza, pero los más comunes son *mp4*, *avi*, *webm*... A parte de la compresión de la propia codificación, el tamaño de este tipo de datos depende principalmente de dos factores: la cantidad de fotogramas por segundo y la resolución de los fotogramas.

### **Fotogramas por segundo – FPS**

Los fotogramas por segundo son una medida esencial en los vídeos que determina la cantidad de fotogramas que hay por segundo de vídeo. Esta medida está muy relacionada con la finalidad del vídeo, ya que no se necesitan los mismos fotogramas por segundo para una película, que suele ser 24 fps [2], que para ver un documental de deporte. Cabe destacar, que a la hora de visualizar un vídeo es importante la tasa de refresco del monitor, para que se pueda apreciar correctamente la cantidad de fotogramas por segundo del vídeo se necesita una tasa de refresco igual o superior. Esto quiere decir que si un vídeo está a 60 fps lo recomendable para ver correctamente el vídeo es que el monitor tenga al menos 60Hz de tasa de refresco.

## Calidad o resolución

La calidad de las imágenes del vídeo es otro de los parámetros más importantes, ya que dependiendo de este valor el vídeo pesará más o menos o se verá mejor o peor. Actualmente la proporción más normal de pantalla es 16:9, las calidades más comunes para esta proporción son:

- 480p → 854 x 480.
- 720p HD → 1280 x 720.
- 1080 p Full HD → 1920 x 1080 .
- 1440p QHD → 2560 x 1440.

## 3.2. Visión por computador

La visión por computador, también llamada visión artificial, es la parte de la ciencia de la computación orientada a la recogida y al tratamiento de imágenes y vídeos [28, 19] para la obtención de información inherente a estos tipos de datos.

Por la tipología de los datos con los que se trabaja en la visión por computador, el procesamiento de éstos es muy costoso. El elevado coste computacional es causado en gran parte por la calidad de las imágenes y en el caso de los vídeos, además de la calidad de la imagen, por la cantidad de fotogramas por segundo con el que ha sido grabado.

### Estimación de la posición o postura

Una de las principales funciones que tiene la visión artificial es la detección de movimiento, la cual consiste en primero detectar la posición de cada una de las partes del cuerpo (depende del propio algoritmo la división que se quiera realizar sobre el cuerpo) para posteriormente realizar un seguimiento de estos elementos. Un ejemplo de este tipo de funcionalidad se puede ver en la figura 3.1a.

### Segmentación

Es otra de las funciones más usadas de la visión por computador, en ella se busca analizar la imagen para poder detectar y delimitar el espacio los distintos objetos, personas o animales que hay en la imagen, como se puede

ver en el ejemplo de la figura 3.1b. Este espacio delimitado se obtiene de dos formas principalmente:

- Figura exacta, se delimita la silueta del objeto, persona o animal detectado con un conjunto de puntos que representan la silueta de éste.
- Forma geométrica, es el tipo más usado debido a que requiere menor capacidad de cómputo. La forma más utilizada es el rectángulo, pero se podrían usar otro tipo de formas para delimitar y segmentar el objeto, persona o animal detectado.



(a) Ejemplo estimación de la postura.



(b) Ejemplo segmentación.

Figura 3.1: Ejemplos tipos de utilidades de la visión por computador.

### 3.3. Redes neuronales

Son modelos basados en la analogía con la corteza cerebral de los mamíferos, aunque no están cerca, al menos de momento, de ser tan complejos como el cerebro biológico. El funcionamiento de estos modelos es relativamente sencillo, ya que «solo» son un conjunto de unidades básicas (también llamadas neuronas), que reciben una entrada y devuelven una salida. Donde realmente reside su complejidad, al igual que en el cerebro humano, es en las conexiones de estas unidades básicas para formar un modelo complejo capaz de realizar multitud de tareas [19].

Estas unidades interactúan con los datos que les llegan con una función de activación que puede excitar a esta unidad o neurona y así pasar la información modificada a sus conexiones. Además, las distintas conexiones entre las unidades tienen un peso definido, que expresa la fuerza del enlace.

Este peso se modifica a lo largo del entrenamiento de la red para poder obtener mejores resultados.

Según como se propague la información por la red, estas pueden dividirse en dos tipos:

- **Feed-forward networks:** redes neuronales donde la información solo avanza por ésta hacia delante, no existe ningún tipo de bucle entre las conexiones de las unidades que haga pasar la información dos o más veces por una de éstas. En este tipo de redes neuronales se encuentran el perceptrón multicapa y las redes neuronales convolucionales
- **Feed-back networks:** redes neuronales que sí admiten bucles entre sus conexiones, lo que les permite tener capacidad de memorización. Alguno ejemplos de este tipo de redes son las redes neuronales recurrentes o las *Long-Short Term Memory* [32].

## Redes neuronales convolucionales

Las redes neuronales convolucionales o CNNs [19], de sus siglas en inglés Convolutional Neural Network, son de los tipos de redes más usados en la actualidad, sobre todo cuando se trabaja con datos de alta dimensionalidad, principalmente imágenes y vídeos. Las redes neuronales convolucionales tienen dos fases (figura 3.2):

1. Fase de extracción de características.
2. Fase de clasificación, puede realizarse con un perceptrón multicapa por ejemplo.

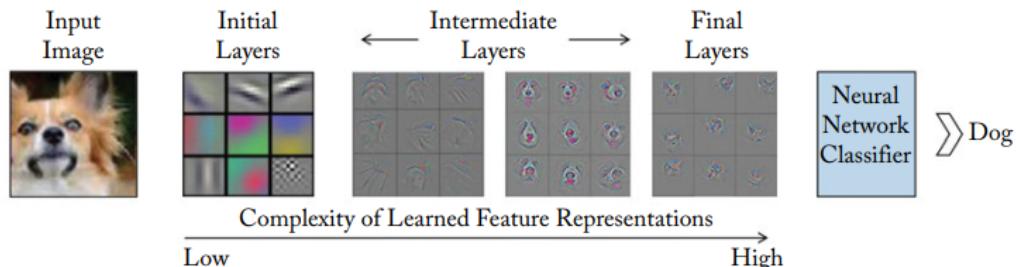


Figura 3.2: Organización de una red neuronal convolucional [19].

Cada fase está relacionada con un paradigma de aprendizaje distinto, ya que en la extracción de características a partir de filtros se realiza de

forma no supervisada, mientras que en la fase de clasificación se realiza un aprendizaje supervisado a partir de los datos obtenidos en la fase de extracción de características.

Aun así, el primer paso que siempre se ha de llevar a cabo antes de entrar en las distintas capas es realizar un preprocesado de la imagen. Éste puede ser por:

- Extracción de la media, para a partir de ésta poder centrar los datos. Para cada dato de entrenamiento  $x$  de un conjunto  $N$  tal que  $x \in \mathbb{R}^{h \times w \times c}$ , donde  $h$  es el alto de la imagen,  $w$  es el ancho y  $c$  es color (normalmente 3 dimensiones, RGB), se puede definir la extracción de la media con la siguiente fórmula:

$$x' = x - \hat{x}, \hat{x} = 1/N \sum_{i=1}^N x_i \quad (3.1)$$

- Normalización. Con la fórmula:

$$x'' = \frac{x'}{\sqrt{\frac{\sum_{i=1}^N (x_i - \hat{x})^2}{N-1}}} \quad (3.2)$$

- Blanqueo con PCA para reducir las correlaciones entre las diferentes dimensiones de los datos a partir de una normalización independiente para cada una de ellas.
- *Local Contrast Normalization*, tipo de normalización que permite resaltar los valores más elevados.

Los tipos de preprocesados que se suelen realizar son las extracción de la media y la normalización, que a la vez son los métodos más sencillos.

## Tipos de capas en una red neuronal convolucional

### Capas convolucionales

Como ya se ha comentado, la funcionalidad de estas capas es la extracción de características a partir de filtros, pero ¿qué es un filtro? Un filtro es una malla con valores discretos, normalmente cuadrada, que representan pesos que se modifican en la fase de entrenamiento de la red. Es con cada uno de estos filtros con los que se recorren los datos para obtener como resultado un resumen de los datos de entrada. Como se ve en la figura 3.3, el resumen se

calcula aplicando el filtro a los datos multiplicando los valores de los datos por su posición en el filtro y luego sumando los valores.

Al aplicar el filtro hay que tener en cuenta el *stride*, que es el movimiento que hace el filtro en cada paso, en la imagen se puede ver como este valor es de 1 por lo que la imagen avanza horizontal y verticalmente de uno en uno. Valores más altos de *stride* producen resúmenes más comprimidos pero puede que menos precisos, lo que se suele llamar un submuestreo de los datos.

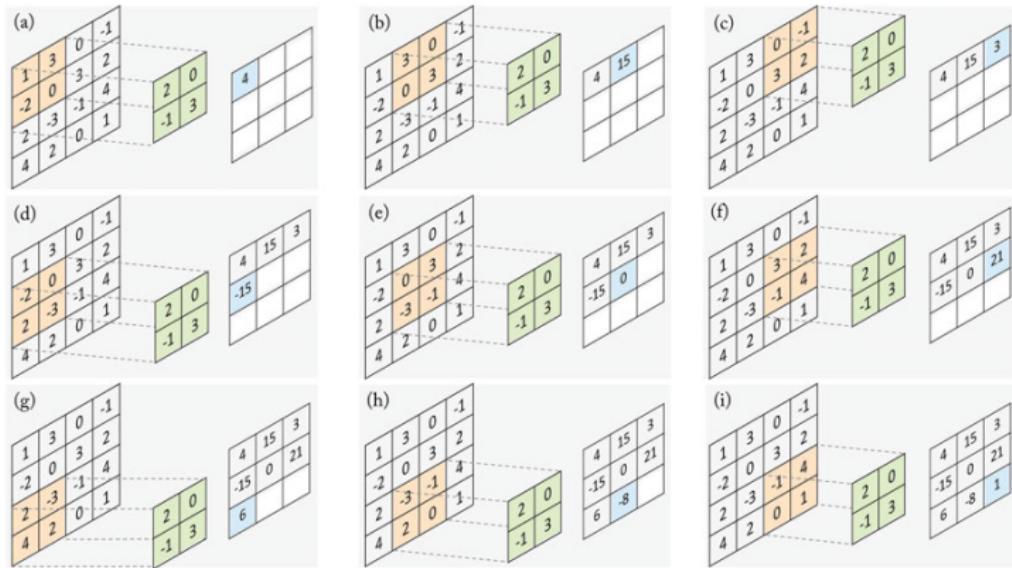


Figura 3.3: Ejemplo aplicación de un filtro con *stride* 1 [19].

Para calcular el tamaño de los datos una vez pasado un filtro se puede usar la siguiente fórmula, donde  $h$  es *height* (alto),  $w$  es *width* (ancho),  $s$  es el *stride* y  $f$  es el tamaño del filtro:

$$h' = \left\lfloor \frac{h - f + s}{s} \right\rfloor; w' = \left\lfloor \frac{w - f + s}{s} \right\rfloor \quad (3.3)$$

Siendo  $\lfloor \cdot \rfloor$  la función de parte entera o *floor operation*. Si se aplica esta fórmula a los datos de la imagen 3.3:

$$h' = \left\lfloor \frac{4 - 2 + 1}{1} \right\rfloor = 3; w' = \left\lfloor \frac{4 - 2 + 1}{1} \right\rfloor = 3 \quad (3.4)$$

Con este tipo de capas existe un problema, ya que hay muchas situaciones que necesitan que se conserve el tamaño de los datos, como por ejemplo en

una segmentación, pero si se utiliza este tipo de capas aunque se use el valor mínimo, *stride* 1, se siguen perdiendo datos suavizando sobre todo los bordes. Es por ello que surgió el concepto *zero-padding* que permite mantener el tamaño original de los datos.

El *zero-padding* consiste aadir a los datos de entrada bordes con valores 0 para que cuando se pase el filtro el tamaño de los datos se intente mantener constante. Con el uso del *zero-padding* la fórmula se modifica usando  $p$  como el número de aumentos con valores a 0 en cada dimensión [9]:

$$h' = \left\lfloor \frac{h - f + s + p}{s} \right\rfloor; w' = \left\lfloor \frac{w - f + s + p}{s} \right\rfloor \quad (3.5)$$

El uso de *padding* se puede dividir en 3 categorías:

- *Valid Convolution*: es el caso más sencillo, el filtro siempre se mantiene en posiciones válidas. Las dimensiones finales se reducen tanto en el alto ( $h$ ) como en ancho ( $w$ ) en el tamaño del filtro ( $f$ ) menos 1.
- *Same Convolution*: en este caso los datos de entrada y de salida de la capa tienen el mismo tamaño, para conseguirlo se ha de realizar *zero-padding* correctamente, ya que no se puede dar en todos los casos.
- *Full Convolution*: en este caso se utiliza el mayor *padding* posible, que se consigue cuando solo un elemento de los datos de entrada está involucrado en todas las operaciones con el filtro.

Como se puede ver, el *stride* y sobre todo el tamaño del filtro a aplicar es muy importante en este tipo de capas, es por ello que definir un buen valor para estos puede ser esencial en el desarrollo del modelo. Por esta razón existe la función de *Receptive Field* que permite descubrir el valor para el tamaño de los filtros (todos los filtros de todas las capas del mismo tamaño) más efectivo. La fórmula del *Effective Receptive Field* para la capa  $N$  de las  $N$  capas convolucionales es:

$$RF_{eff}^n = f + n * (f - 1) \quad (3.6)$$

Pero, como se ha visto, los filtros no suelen tener el mismo tamaño ni el mismo *stride*, es por ello que la función de *Receptive Field* para el tamaño de más de un filtro es:

$$RF_{eff}^n = RF_{eff}^{n-1} + ((f_n - 1) * \prod_{i=1}^{n-1} s_i) \quad (3.7)$$

Donde  $f_n$  es el tamaño del filtro de la capa  $n$ ,  $s_i$  es el valor de *stride* para la capa  $i$ , y  $RF_{eff}^{n-1}$  representa el *effective receptive field* de la capa anterior.

Como ya se ha comentado, uno de los principales problemas de las capas convolucionales son los valores de los bordes, ya que al aparecer menos en las operaciones de los bordes se ven menos representados en las salidas de las capas. Es por ello que se propuso una modificación de las capas convolucionales llamada *Dilated Convolution*, la cual tiene una dilatación ( $d$ ) siendo esta la distancia entre los valores del filtro a tener en cuenta. Como se puede ver en la figura 3.4, donde se obtiene un filtro de tamaño 3 y con una dilatación de 2, los valores de los bordes tienen un mayor peso en la salida de la capa.

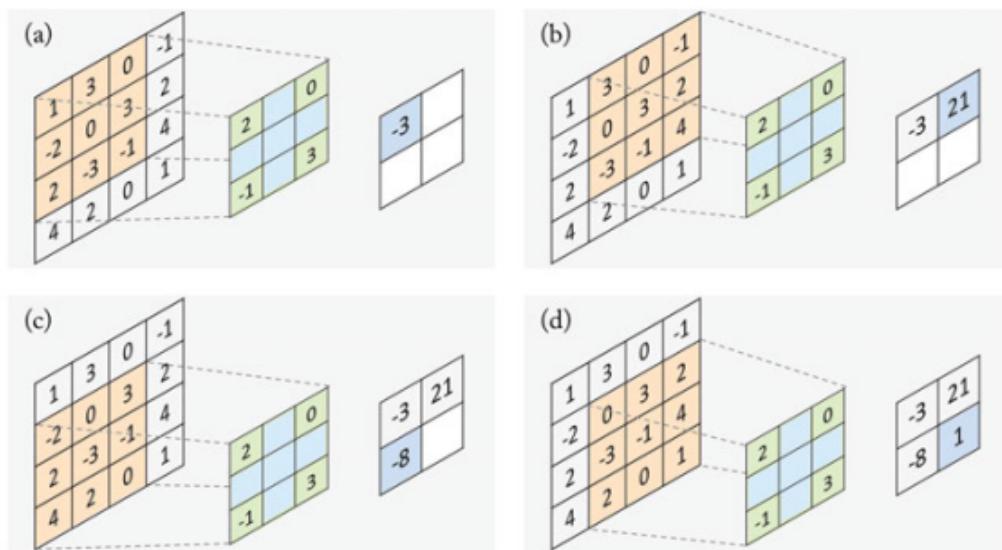


Figura 3.4: *Dilated Convolution* con dilatación a 2 [19].

Al haber añadido un nuevo parámetro, el tamaño de la salida viene dado por las siguientes fórmulas:

$$\begin{aligned} h' &= \frac{h - f - (d - 1) * (f - 1) + s + 2p}{s} \\ w' &= \frac{w - f - (d - 1) * (f - 1) + s + 2p}{s} \end{aligned} \quad (3.8)$$

### Capas de agrupación

Tipo de capa en la cual se realiza una agrupación por un número fijo de datos. Esta agrupación se puede realizar de diversas formas, como puede

ser la media, el mínimo o el máximo, entre otros. Como en las capas convolucionales, se ha de pasar el valor del *stride* para indicar cómo va a ser el movimiento del segmento de agrupación. Además, se le ha de pasar el tamaño del segmento de agrupación, algo parecido a lo que pasaba con el tamaño del filtro, pero en este caso no hay valores en el filtro, simplemente se agrupan siguiendo una función los datos del segmento. Un ejemplo se puede ver en la figura 3.5, donde se puede observar como los valores se van agrupando acorde con el valor máximo del segmento o bloque comprobado. Los tamaños de las salida de los datos tras pasar por una capa se pueden calcular con las siguientes fórmulas, donde  $f$  pasa a ser el tamaño del segmento o bloque:

$$h' = \left\lceil \frac{h - f + s}{s} \right\rceil; w' = \left\lceil \frac{w - f + s}{s} \right\rceil \quad (3.9)$$

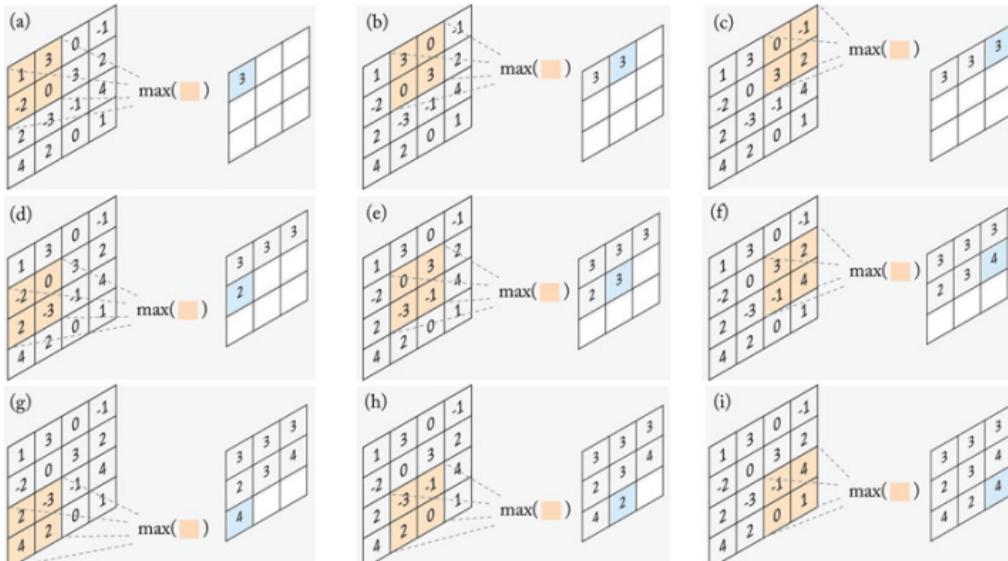


Figura 3.5: Ejemplo de capa de agrupación de tamaño 2 [19].

Este tipo de capas son muy útiles para compactar la información, para detectar cambios en los datos como puede ser para el movimiento o la posición en una imagen.

### Capas totalmente conectadas

Tipo de capa en la cual todas sus unidades están conectadas con todas las unidades de la capa anterior. En este tipo de capas se realiza una operación

parecida a las capas convolucionales, salvo que en este caso se realiza con una tamaño de filtro de 1, es decir, se realiza la operación dato a dato.

### Capas de no linealidad

Capas en las cuales la función de activación de cada unidad toma como valor de entrada un número y da como salida un valor en un rango corto, como puede ser  $[0,1]$  o  $[-1,1]$ . Este tipo de capas suele ir después de las capas «pesadas» como las capas convolucionales o de las capas totalmente conectadas. Este tipo de capa es muy importantes utilizarlas después de las capas «pesadas» debido a que permite mapear espacios no lineales. Algunas de las funciones de activación de esta capa más comunes se pueden ver en la figura 3.6.

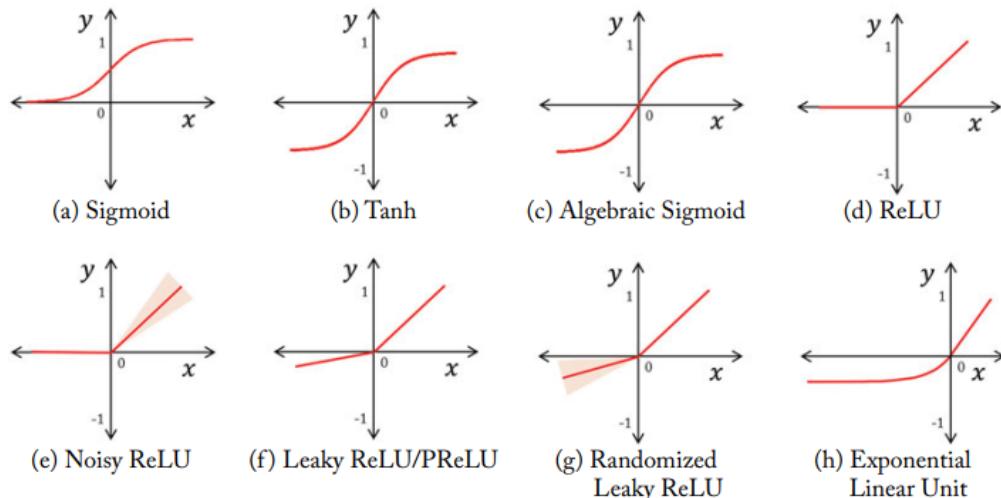


Figura 3.6: Funciones de activación más comunes en capas de no linealidad [19].

## ResNet

ResNet o *Residual Network* es un tipo de arquitectura de redes neuronales convolucionales creada por Microsoft [12]. Este tipo de CNN se caracterizan por saltarse las conexiones de identidad en los bloques residuales, para así obtener un modelo mucho más sencillo de entrenar. En un bloque residual, los datos de entrada se duplican, pasando unos por las transformaciones y otros por las conexiones de identidad, estas conexiones, como se puede ver en la figura 3.7, sirven para saltarse las capas de transformación, pasando los datos iniciales tal cual llegan.

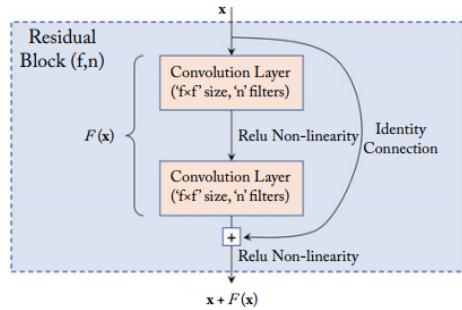


Figura 3.7: Ejemplo bloque residual en ResNet [19].

## Region-Based CNN

Las *Region-Based Convolutional Neural Networks* o RCNN son un tipo de redes neuronales convolucionales orientadas al reconocimiento de objetos en imágenes. Este tipo de redes convolucionales, como se puede observar en la figura 3.8, tiene 3 módulos [19]:

- Módulo de extracción de propuestas a región.
- Módulo de extracción de características. Uso de redes neuronales convolucionales para la extracción de características de cada región propuesta.
- Módulo de clasificación de regiones. Se entrena un SVM (Support Vector Machine) por cada clase a predecir, después se usa para clasificar las regiones a partir de las características extraídas en el módulo anterior.

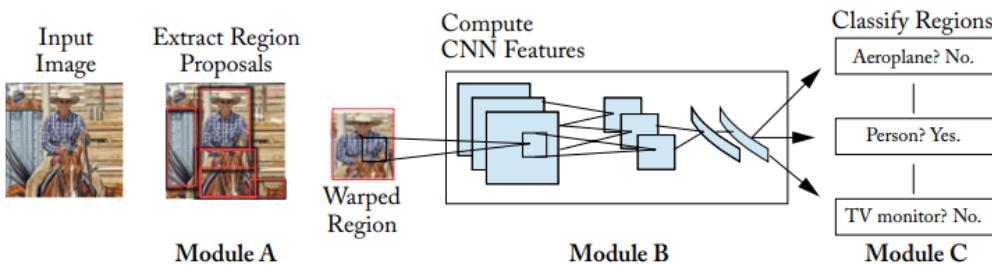


Figura 3.8: Region-Based CNN [19].

## FPN

FPN o Feature Pyramid Network es un tipo de red neuronal piramidal que se utiliza junto con las redes neuronales convolucionales para tareas relacionadas con las imágenes, tales como la detección de objetos (usándose junto una RCNN) o la segmentación. Este tipo de redes, como se puede observar en la figura 3.9, tiene dos fases [21]:

- *Bottom-up pathway*: Fase en la cual se utiliza la red neuronal convolucional para extraer características de la imagen, como por ejemplo con ResNet. Cada capa dentro de esta red neuronal convolucional es de la mitad del tamaño que la anterior (por lo que hay que tener en cuenta todos los atributos de las capas de una red neuronal ya comentados). En el ejemplo de ResNet se obtiene la salida de cada capa como el resultado de cada bloque residual.
- *Top-down pathway and lateral connections*: En esta fase, en el primer nivel, se toma como entrada la salida de la última capa de la fase anterior, y se va bajando de nivel pasando al siguiente nivel la predicción del anterior y la salida de la capa de la red neuronal que está al mismo nivel.

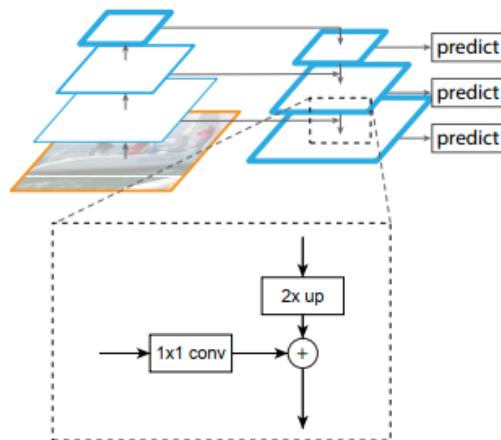


Figura 3.9: Feature Pyramid Network [21].

## 3.4. Puntos y vectores

El punto es la representación mínima en una espacio, donde representa una posición exacta dentro de este espacio. Un punto dentro de un espacio

se representa con tantos valores como dimensiones tiene el espacio donde se encuentra.

El vector es otro de los conceptos básicos matemáticos que se define como un segmento orientado. Un vector se representa a partir de 3 valores, su módulo o longitud del segmento, su dirección y su sentido.

## Distancia entre puntos

La distancia entre dos puntos es el espacio que los separa. Esta distancia se puede calcular de distintas formas, ya que depende mucho del espacio en el que se encuentren los puntos. Algunas de las distancias más comunes son:

- **Manhattan:** distancia que se calcula como la suma de las diferencia de los valores de las distintas dimensiones de los puntos.
- **Euclídea:** distancia que se calcula como la longitud de la línea que une los dos puntos, a partir del teorema de Pitágoras.

## Ángulo entre vectores

Ángulo mínimo que hay que girar un vector para conseguir la dirección del otro vector con el cual ha de compartir al menos un punto [25].



---

# Técnicas y herramientas

---

En este apartado se van a comentar las distintas técnicas y herramientas utilizadas en el desarrollo del proyecto.

## 4.1. Visión por computador ~ Detección de movimiento

Existen distintos algoritmos de detección de movimiento, cada uno diferenciado por el uso de distintos lenguajes de programación y diferentes métodos de inteligencia artificial con los que entrenar el modelo. Como en este proyecto se ha querido trabajar en *Python*, se han investigado los siguientes algoritmos que están implementados en este lenguaje de programación.

### Detectron

*Detectron* es un proyecto de inteligencia artificial de *Facebook* centrado en la detección de objetos y personas con el uso de algoritmos de *deep learning* a partir de redes neuronales convolucionales [15]. Este proyecto se apoya en *PyTorch*, una de las librerías de inteligencia artificial más usadas en *Python*.

El proyecto cuenta ya con una segunda versión llamada *Detectron2*, en la cual se han mejorado y añadido más modelos [30]. La mayoría de modelos de *Detectron2* usan redes neuronales convolucionales.

Como se explicará en el apartado 5.2, esta herramienta fue la elegida para la obtención de la posición o postura del paciente.

## PoseNet

PoseNet es un proyecto de *Google* orientado únicamente al seguimiento del movimiento de las personas centrando su análisis en los puntos claves del cuerpo humano. El proyecto se basa en la librería de inteligencia artificial de la propia compañía *Google* llamada *TensorFlow* [1].

## 4.2. Estación de trabajo

El trabajo con vídeos y la visión por computador necesita gran cantidad de capacidad de cálculo. Es por ello que la realización del proyecto en un ordenador personal es casi imposible, o al menos ralentizaría mucho el avance de éste. Es por esta razón que se ha decidido utilizar para el desarrollo del proyecto, sobre todo para la parte de ejecución de los algoritmos, una estación de trabajo de la Universidad de Burgos. Esta estación de trabajo, que se llama Gamma, se encuentra en uno de los despachos del departamento de Ingeniería Informática en la Escuela Politécnica Superior. Las especificaciones de esta estación de trabajo son:

- 3 Nvidia Titan XP, con 12 GB de memoria cada una.
- Intel Xeon, con 16 núcleos.
- 128GB de RAM.

---

# **Aspectos relevantes del desarrollo del proyecto**

---

En este apartado se va a comentar, a manera de resumen temporal, el desarrollo del proyecto. Es en este apartado donde se comentarán las opciones y decisiones tomadas, los problemas surgidos y todos los aspectos importantes. Se ha considerado que la mejor forma de organizar este apartado es en secciones donde se comenta cada apartado del desarrollo.

## **5.1. Desarrollo FIS-HUBU**

FIS-HUBU es proyecto que permite realizar vídeo llamadas entre responsables y pacientes con Parkinson para realizar rehabilitaciones de manera *online*, es decir, sin la necesidad de desplazarse hasta la consulta o el hospital. Este proyecto se ha desarrollado junto con la investigación en la universidad Carlos III, con el proyecto «Fundación Burgos por la investigación de la salud complejo asistencial universitario de Burgos PI19/00670. Estudio de factibilidad y coste-efectividad del uso telemedicina con un equipo multidisciplinar para enfermedad de Parkinson».

Esta aplicación, que ha sido desarrollada junto con mi compañero José Luis Garrido Labrador, permite por parte del responsable observar y evaluar la evolución del estado de un paciente, esta evolución también es visible para el paciente que puede ver su propio progreso.

La aplicación puede dividirse en distintas partes que serán comentadas a continuación.

## Vídeo llamadas

El punto principal de la aplicación, y por ende su principal uso, son las vídeo llamadas entre pacientes y responsables o terapeutas que permitan sustituir la rehabilitaciones presenciales en consulta por rehabilitaciones *online*, permitiendo así que estas se puedan dar más a menudo y puedan ser accesibles para un mayor número de personas, sobre todo para aquellos pacientes que no se pueden desplazar o viven aislados geográficamente.

Primero se realizó una investigación sobre las principales plataformas de vídeo llamadas. Lo que se estaba buscando de estas plataformas era:

- Creación de llamadas de manera sencilla y automática. Si es posible a partir de url.
- Vídeo llamada estable sin necesidad de una gran conexión.
- Plataforma que permita grabar la cámara de los pacientes.
- Plataforma gratuita.

Dentro de las plataformas que se investigaron están las más conocidas aplicaciones de este tipo como puede ser *Skype*, pero al final se decidió utilizar *Jitsi*<sup>1</sup> ya que proporciona todas las necesidades anteriormente comentadas, y además permite en un futuro poder crear un servidor propio donde poder modificar parámetros como la calidad de las llamadas.

## Responsable

Parte de la aplicación donde los responsables pueden realizar las siguientes tareas:

- Iniciar una vídeo llamada con un paciente.
- Evaluar la evolución de los pacientes.
- Modificar las evaluaciones de los pacientes.
- Comprobar la evolución de los pacientes.

La interfaz de la aplicación para tipo de usuario es sencilla y clara, como se puede ver en el menú principal, figura 5.10, o en el menú de estadísticas, figuras 5.11 y 5.12.

---

<sup>1</sup>Jitsi: <https://jitsi.org/>



Figura 5.10: Menú principal de un responsable.



Figura 5.11: Menú de estadísticas.

## Paciente

Los pacientes que van a utilizar la aplicación, son pacientes de edad avanzada con Parkinson, y muchos de un estadio avanzado. Es por ello que el diseño de la aplicación se orientó principalmente en la sencillez y facilidad de uso para estos pacientes.

Para poder realizar una aplicación lo más accesible posible primero se ha de saber la forma que van a tener los pacientes de interactuar con ésta. En este caso los pacientes van a utilizar un mando de SNES<sup>2</sup> con botones de colores, es por ello que se ha aprovechado estos colores para poder mostrar en la interfaz de la aplicación el botón que han de pulsar para realizar la acción. Además, se ha creado un botón de ayuda que carga una página donde se puede ver la acción que realiza cada botón del mando. Un ejemplo de la interfaz de esta aplicación se puede ver en la figura 5.13.

## Dispositivos necesarios

Para poder utilizar la aplicación se necesitan distintos dispositivos, que dependiendo del tipo de usuario (responsable o paciente) son unos u otros.

<sup>2</sup>SNES: *Super Nintendo Entertainment System*.



Figura 5.12: Ejemplo de la evolución de un paciente vista por un responsable.

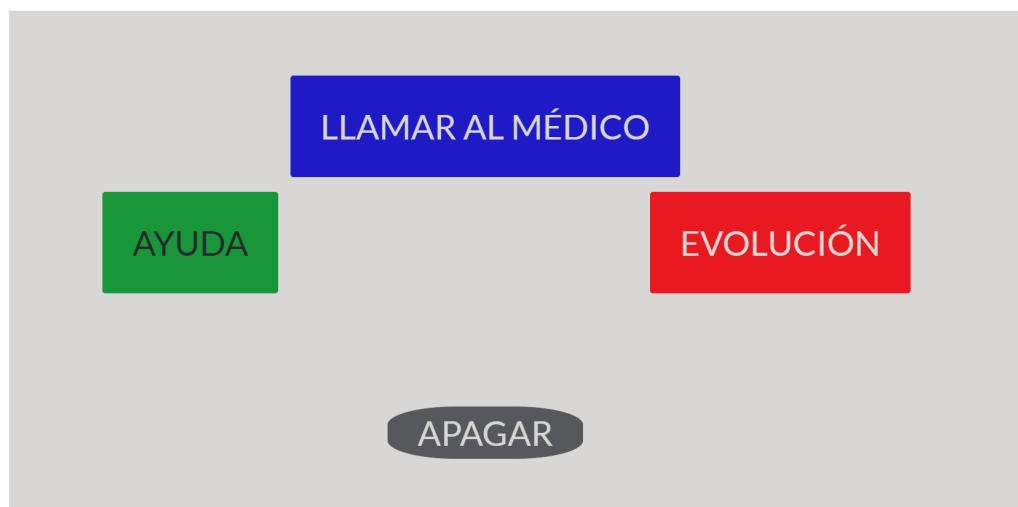


Figura 5.13: Menú principal de los pacientes.

Para los responsables lo único que se necesita es un ordenador con conexión a *Internet* y una cámara conectada a éste.

Por otro lado, se presupone que los pacientes no disponen de ningún dispositivo capaz de cargar la aplicación, es por ello que a cada paciente se le proporciona:

- Dispositivo: MSI - Cubi N 8GL-001BEU N4000 1,10 GHz Western

Digital - Green M.2 120 GB Serial ATA III.

- Cámara: Logitech HD Pro WebCam C920.

## Conexión

Como ya se ha comentado, el principal uso de la aplicación es la realización de rehabilitación *online* para pacientes, mayoritariamente de tercera edad, que no se pueden desplazar a las consultas. Muchas de estas personas mayores viven en lugares donde no se tiene contratada ninguna línea de *Internet*, es por ello que además del desarrollo de la aplicación se contrataron una serie de *routers* 4G para poder proporcionar conexión a los dispositivos necesarios.

## 5.2. Investigación algoritmos de visión por computador

Tras haber desarrollado la versión inicial, donde en un futuro se quiere añadir los resultados de este proyecto, se pasó a realizar la primera investigación sobre los distintos algoritmos de visión por computador capaces de detectar y seguir los movimientos de una persona.

De estos algoritmos se necesita:

- Posibilidad de cargar un modelo existente o crear un modelo capaz de detectar a la persona que sale en la imagen.
- Posibilidad de cargar un modelo existente o crear un modelo capaz de detectar la posición de la persona.
- Que el procesado de nuevos fotogramas para detectar a la persona y su posición se realice en poco tiempo.
- Que la salida del procesado de los fotogramas pueda servir para una posterior comparación con el ejercicio base.

Teniendo todos estos factores en cuenta, se estudió qué herramientas se pueden usar para esta fase del trabajo. Se buscaron herramientas programadas en *Python* para poder conectarse bien con el resto del proyecto y porque es uno de los lenguajes con los cuales se tiene más soltura, tanto por parte del alumno como por parte de los tutores para resolver dudas y ayudar en los problemas. Las herramientas encontradas fueron:

- *TF-Pose-Estimator.*
- *PoseNet.*
- *Detectron2.*

De cada una de estas herramientas se realizó una investigación y experimentación para evaluar si cumplían las necesidades requeridas. Al finalizar esta etapa, la única herramienta que permitía realizar todas las necesidades era *Detectron2*. Además, permite con un modelo ya creado por los propios desarrolladores realizar las tareas de predicción de elementos y de la posición de la persona.

Cabe destacar uno de los grandes problemas que surgió en esta fase, y fue justamente con *Detectron2*, la herramienta elegida. El problema era que las versiones de *CUDA* y de *PyTorch* no eran compatibles. El problema se agrandó al estar trabajando en un *workstation* de la universidad como es *Gamma* que utilizan otros investigadores y también por la compleja estructura del propio computador. Tras hablar con el administrador de la computadora, que es uno de los tutores de este trabajo, el doctor Álvar Arnaiz González, se pudo arreglar el problema actualizando la versión de los *drivers* de *CUDA*, y una vez actualizados descargando la versión compatible de *PyTorch*.

### 5.3. Investigación de *Detectron2*

La fase de investigación de *Detectron2* fue una de las más importantes, y por ende, más costosas en tiempo. Fue en esta fase donde se investigaron las distintas formas que tiene la herramienta para crear o importar modelos con los que poder trabajar. Una vez se seleccionó el modelo se realizó otro estudio para ver qué configuración era la más correcta en el problema tratado.

Al ser una de las fases en las que más tiempo se ha dedicado, también es una de las fases donde surgieron más problemas tanto en la creación de los modelos, como en la visualización y el almacenamiento de los resultados obtenidos. Todos estos problemas serán comentados en este apartado.

#### Selección del modelo

Tras haber elegido a *Detectron2* como herramienta de visión por computador para detectar a los pacientes y obtener de estos sus posiciones, había

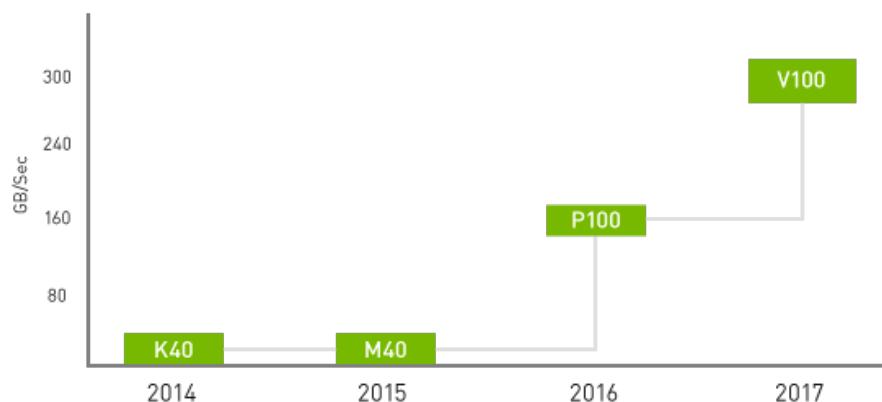
que elegir qué modelo, dentro de las posibilidades de la herramienta, utilizar para realizar estas tareas.

*Detectron2* permite dos formas de obtener un modelo:

- Crear un modelo propio a partir de una gran cantidad de datos.
- Importación de modelos ya entrenados.

Ante la falta de datos para la creación de un modelo que diese buenos resultados, y al comprobar en la fase anterior que con la importación de modelos existentes se podían obtener buenos resultados, la investigación se centró en los modelos ya existentes entrenados por los creadores de la herramienta. Estos modelos de redes neuronales fueron entrenados en servidores *Big Basin* de *Facebook*, sucesores de los servidores *Big Sur*, ambos orientados al uso de arquitecturas con varias GPUs potentes para el entrenamiento y uso de algoritmos de inteligencia artificial [11]. En concreto, estos modelos fueron entrenados con 8 Nvidia Tesla V100 con *NVLink*, una tecnología que mejora las interconexiones entre GPUs proporcionando mayor ancho de banda, haciendo el sistema más escalable [24]. La mejora con otras generaciones de comunicación *inter-GPU* se puede ver en la figura 5.14.

**NVLink Performance**



NVLink in Tesla V100 doubles inter-GPU communication bandwidth compared to the previous generation, so researchers can use larger, more sophisticated applications to solve more complex problems.

Figura 5.14: Mejora del rendimiento en GB/s con NVlink [24].

Los modelos creados en *Detectron2* se diferencian en los siguientes tipos:

- COCO Detection with Faster R-CNN (Figura 5.15a): modelo que predice los objetos y personas de la imagen.
- COCO Detection with RetinaNet (Figura 5.15b): modelo que detecta objetos y personas. Como se puede ver en la imagen de ejemplo, la interpretación no es buena aun utilizando un *threshold* alto.



(a) Modelo COCO Detection with Faster R-CNN, `faster_rcnn_R_50_C4_1x`. (b) Modelo COCO Detection with RetinaNet, `retinanet_R_101_FPN_3x`.

Figura 5.15: Modelos *Detectron2*.

- COCO Detection with RPN and Fast R-CNN: modelos que no se han conseguido probar ya que no siguen la misma estructura que el resto de modelos.
- COCO Instance Segmentation Baselines with Mask R-CNN (Figura 5.16a): modelos de detección de objetos y persona, con delimitación en el área ocupada.
- COCO Person Keypoint Detection Baselines with Keypoint R-CNN (Figura 5.16b): modelos que detectan a las posiciones y unos puntos claves de ellas, estos puntos son nariz, ojos, orejas, hombros, codos, muñecas, cadera, rodilla y tobillos. Como se puede ver en la imagen de ejemplo, estos modelos predicen bastante bien la posición, pero a veces detectan otras cosas como personas, por lo que se posteriormente se vio necesario un estudio sobre el *threshold*.
- COCO Panoptic Segmentation Baselines with Panoptic FPN (Figura 5.17a): modelos de segmentación como COCO Instance Segmentation Baselines with Mask R-CNN, obtienen resultados muy parecidos.
- LVIS Instance Segmentation Baselines with Mask R-CNN (Figura 5.17b): modelos orientados a la detección de objetos, necesitan un *threshold* bajo para detectar diversos objetos.



(a) Modelo COCO Instance Segmentation Baselines with Mask R-CNN, mask\_rcnn\_R\_50\_DC5\_3x. (b) Modelo COCO Person Keypoint Detection Baselines with Keypoint R-CNN, keypoint\_rcnn\_R\_101\_FPN\_3x.

Figura 5.16: Modelos Detectron2.



(a) Modelo COCO Panoptic Segmentation Baselines with Panoptic FPN, panoptic\_fpn\_R\_101\_3x. (b) Modelo LVIS Instance Segmentation Baselines with Mask R-CNN, mask\_rcnn\_X\_101\_32x8d\_FPN\_1x.

Figura 5.17: Modelos Detectron2.

- Por último se encuentran modelos como Cityscapes & Pascal VOC Baselines, otras configuraciones y algoritmos de la primera versión de la herramienta Detectron. Estos modelos son muy parecidos a los ya mostrados.

Con el estudio de todos los modelos existentes en Detectron2, se ha comprobado como los únicos modelos que sirven para el problema planteado son COCO Person Keypoint Detection Baselines with Keypoint R-CNN. Dentro de este tipo de modelos existen un total de 4 modelos distintos. Para la elección del mejor de estos modelos se realizó otro estudio, esta vez comprobando los tiempos de ejecución, ya que todos los modelos de este tipo realizan unas buenas predicciones (principalmente porque el problema es sencillo, al tener a la persona en el centro de la imagen).

Para realizar este estudio se analizó el tiempo de procesado e impresión de varios vídeos con cada uno de los modelos. Para ello se calculó estos valores para los 4 posibles modelos con un total de 7 vídeos. De este estudio se obtuvieron los resultados que se pueden ver en la tabla 5.1.

Modelo	T. Carga (ms)	T. Procesamiento (ms)	N. Fotogramas	Periodo
keypoint_rcnn_R_50_FPN_1x	8083,19	216519,97	1989	108,86
keypoint_rcnn_R_50_FPN_3x	7539,65	216625,36	1989	108,91
keypoint_rcnn_R_101_FPN_3x	9806,49	259716,33	1989	130,58
keypoint_rcnn_X_101_32x8d_FPN_3x	14853,03	407086,70	1989	204,67

Tabla 5.1: Tabla con el estudio de los modelos de posición ordenado por ratio.

Como se puede observar, los datos obtenidos para los modelos son muy parecidos, sobre todo para los dos con menor periodo (*T.Proc./Fotogramas*). Tras observar estos datos se decidió elegir al modelo `keypoint_rcnn_R_50_FPN_3x` (es además el modelo con el que se hicieron las primera pruebas y con el que se decidió a *Detectron2* como herramienta para detectar el movimiento) para utilizarlo en el resto del proyecto, debido a que aunque es el segundo con el periodo más bajo, con una diferencia casi inapreciable de 0,05 milisegundos, su tiempo de carga es de medio segundo inferior al modelo con el mejor tiempo de procesamiento por fotograma.

Además, en este estudio se ha podido comprobar que los modelos trabajan correctamente con independencia del tipo de ropa que lleve la persona o si pasa un objeto por medio de la imagen (normalmente la posición es correcta, pero como se comentará más adelante a veces algunos fotogramas no son correctos), como se pueden ver en la figuras 5.18a y 5.18b.

Se puede comentar que el modelo seleccionado usa una red neuronal piramidal de tipo FPN con ResNet y RCNN para la detección de la persona, y dentro de la persona sus puntos más importantes.

## Definición del *threshold*

Como se ha comentado en el apartado anterior, es necesario definir un *threshold*, que es un valor entre 0 y 1 que define la sensibilidad del modelo al detectar, es decir, con valores cercanos a 0 el modelo tiende a detectar más elementos (muchos de ellos erróneos) y cuanto más cercano a 1 menos sensible es, solo detectando los elementos más claros. Sobre este valor se ha realizado un estudio de posibles valores, los valores probados fueron 0,3, 0,5, 0,75 y 0,99.



(a) Prueba con chaqueta.

(b) Prueba con un objeto de por medio.

Figura 5.18: Pruebas con ropa y objetos.

Con valores bajos del *threshold* se observa que se detectan cosas que no son personas, o se detectan personas que no salen completas en la imagen, como se puede observar en el figura 5.19a. Esto ocurre con los valores 0.3, 0.5 y 0.75 que dan la misma salida. Sin embargo, con el valor 0,99, como se puede ver en la figura 5.19b, se obtienen muy buenos resultados detectando únicamente a la persona que aparece en el centro de la imagen, con la posición exacta en todos los puntos.

(a) Prueba con *threshold* a 0,3.(b) Prueba con *threshold* a 0,99.Figura 5.19: Pruebas con el *threshold*.

## Problemas surgidos

Como se ha comentado a lo largo del apartado, han surgido distintos problemas que se pueden resumir en:

- Problema con los modelos COCO Detection with RPN and Fast R-CNN, que al no tener la misma estructura que el resto de modelos de Detectron2 no se han podido probar.

- Se intentó almacenar los vídeos procesados pero, por problemas con *OpenCV*, no se pudo guardar en esta etapa.

## 5.4. Cálculo de características

Una vez se seleccionó el modelo que se iba a utilizar durante todo el proyecto y se determinó el parámetro *threshold* de los modelos, el siguiente paso fue la interpretación de las salidas del modelo. Y después de haber interpretado con el modelo de *Detectron2* una posición ser capaces de calcular características que definan de la mejor manera los datos.

### Interpretación de las predicciones

Si se analiza la salida tras predecir un imagen con el predictor creado (*DefaultPredictor*), se observa que devuelve un diccionario con una única entrada llamada «instances» donde se almacena toda la información. Dentro de la instancia existen los siguientes apartados:

- **pred\_boxes**: Boxes, estructura propia de *Detectron2* donde se almacenan límites de donde entiende que está la persona en un *tensor*, si se ha detectado más de una persona este Boxes estará formado por varios *tensors*.
- **scores**: *tensor* con las probabilidades de que los elementos sean personas. Es este el valor que se compara con el *threshold* para ver si se considera o no como una persona. Si existe más de un valor estos están ordenados de mayor a menor, este es el orden que se sigue en el resto de valores.
- **pred\_classes**: *tensor* con el índice de la clase, en este caso todos los elementos son 0 ya que solo detecta personas, que se identifican con este índice.
- **pred\_keypoints**: *tensor* con todos los puntos clave de cada elemento (persona) detectada. De cada elemento detectado, si la predicción ha sido correcta, se obtienen un total de 17 puntos, con sus valores *x* e *y*. Después de investigar el posicionamiento de estos puntos se puede decir, como se ve en la figura 5.20, que los puntos representan (teniendo en cuenta que se han detectado todos los puntos y que la persona está mirando de frente al objetivo) las partes que se pueden observar en la tabla 5.2.

<b>Id</b>	<b>Parte</b>
0	Nariz
1 y 2	Ojos (izq.y der.)
3 y 4	Orejas (izq.y der.)
5 y 6	Hombros (izq.y der.)
7 y 8	Codos (izq.y der.)
9 y 10	Muñecas-manos (izq.y der.)
11 y 12	Cadera (izq.y der.)
13 y 14	Rodillas (izq.y der.)
15 y 16	Tobillos (izq.y der.)

Tabla 5.2: Tabla con las partes de la posición y su identificador.



Figura 5.20: Puntos clave predichos con el modelo junto con la etiqueta con su orden.

## Clase de posición

Una vez se conocen las salidas de las predicciones, se pueden usar estas para obtener más información de la posición de la persona, que en futuras etapas fue usada para la comparación de posiciones. Por ello se creó una clase que almacena todos los puntos detectados además de realizar cálculos

para extraer más características.

Para poder obtener más información sobre los puntos se crearon una serie de funciones que permiten:

- Cálculo de puntos medios. Este cálculo se realiza para obtener el punto inferior del cuello que se calcula como el punto medio de la recta que une los dos hombros y se utiliza también para calcular el punto central de la cadera a partir de su punto izquierdo y derecho.
- Cálculo de la distancia entre puntos. Este punto se ha realizado sobre todos los pares de puntos unidos. Se pueden utilizar para intuir profundidades.
- Cálculo de los ángulos en grados. Este cálculo se ha realizado en todas las combinaciones posibles (conjunto de 3 puntos consecutivos), ya que se cree que son las características que más información van a aportar debido a que no dependen de ningún otro valor como puede ser la profundidad a la que está el paciente, su altura...

Como se comentó en apartados anteriores, una vez se tienen los puntos, se ha de realizar una serie de comprobaciones para saber si las posiciones son válidas:

- Si existe más de una persona detectada se selecciona la primera, ya que es la que tiene un mayor *score* y por lo tanto el modelo cree que puede ser con mayor probabilidad una persona.
- Si una predicción no dispone de todos los puntos no se tiene en cuenta. En la práctica con las pruebas que se han realizado esto solo sucede cuando se pasa un objeto por ciertos puntos del cuerpo, cuando ocurre estos hay algunos fotogramas, muy pocos, que no detectan todos los puntos.

## 5.5. Primeras versiones comparación de posiciones

Una vez se había comprobado que el sistema de cálculo de posiciones era correcto se pasó a desarrollar la comparación de posiciones. Para esta primera versión de la comparación de posiciones se decidió usar los ángulos calculados a partir de los puntos y puntos medios de las posiciones, ya que

es la única medida calculada que no depende de la posición del usuario grabado.

## División de la comparación

La principal característica, a parte del uso de los ángulos, de la primera versión de la comparación de posiciones fue la división de la comparación en distintas partes para poder dar un peso mayor o menor a cada una de ellas. Esta división se creyó necesaria tras observar los vídeos de ejemplos de los ejercicios que se proporcionaron, ya que había ejercicios donde se centraban en los brazos, otros en las piernas...

La división que se planteó en esta versión fue (Figura 5.21):

- Brazos: en esta parte se encuentran únicamente los ángulos de los codos de ambos brazos (5 y 6).
- Torso: ángulos de la cadera con la columna (7 y 8), ángulos de los hombros (3 y 4) y ángulos del cuello con respecto a los hombros (1 y 2).
- Piernas: ángulos de la cadera con el fémur de cada pierna (9 y 10) y ángulos de cada rodilla (11 y 12).

La comparación de cada una de las partes se realiza calculando la suma de los valores absolutos de las restas de los valores de los ángulos entre las dos posiciones.

## Problema con los ángulos de los brazos

Tras haber implementado la primera versión se realizaron unas pruebas para comprobar su funcionamiento. En estas pruebas se encontró un fallo muy importante, ya que los ángulos que se obtenían eran hasta 180 grados, si se superaba este valor el ángulo se calculaba sobre el otro lado del punto central, es decir, nunca es superior a 180 grados. Es por ello que posiciones como las que se pueden ver en las figuras 5.22a y 5.22b tiene ángulos en ambos codos muy parecidos, aunque su posición sean casi contrarias.

Este error que se descubrió que pasaba en los codos, se encontró también en la comparación de hombros. Un ejemplo de este error en los hombros se puede ver en las figuras 5.23a y 5.23b que obtienen ángulos muy parecidos, aunque sus posiciones sean casi contrarias.

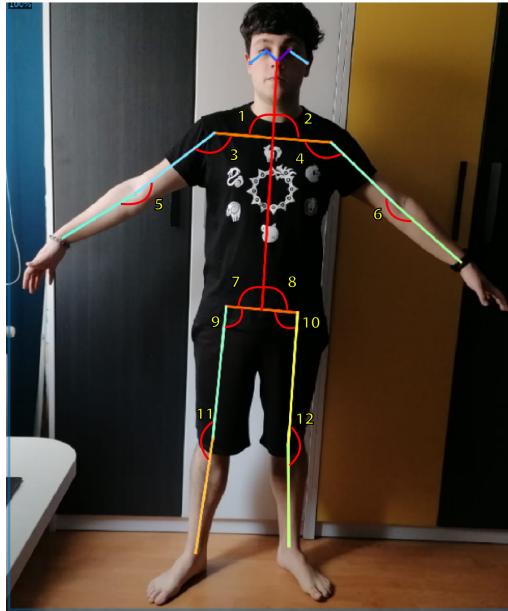
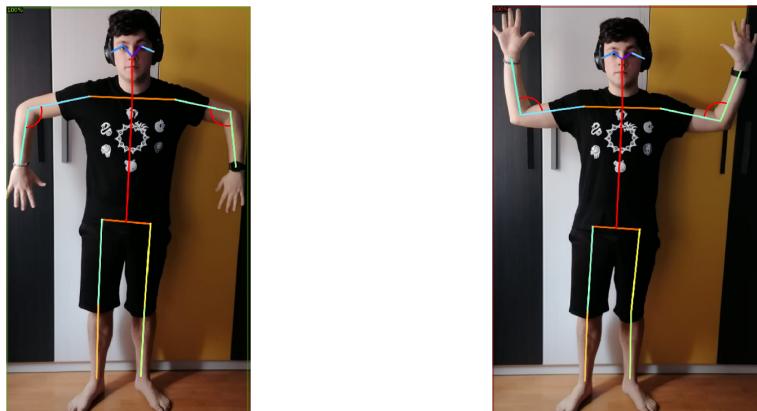


Figura 5.21: Ejemplo ángulos sobre los que se calcula la diferencia entre posiciones.



(a) Posición con los codos en 90 grados mirando hacia abajo. (b) Posición con los codos en 90 grados mirando hacia arriba.

Figura 5.22: Ejemplos problemas con codos.

Para poder solucionar estos problemas lo primero que se hizo fue pasar los ángulos de los hombros que se encontraban en la comparación del torso a la comparación de los brazos. Después, para ambas partes se implementó el algoritmo que se puede ver en la figura 5.24 para poder penalizar estos casos especiales. El algoritmo se recorre por cada una de las zonas y por cada uno



(a) Posición con los hombros en 45 grados mirando hacia abajo. (b) Posición con los hombros en 45 grados mirando hacia arriba.

Figura 5.23: Ejemplos problemas con hombros.

de los lados (izquierda y derecha) para ambas posiciones, y se penaliza solo cuando una zona de un lado tiene un tipo de 1 y la correspondiente en la otra zona tiene tipo 2, o viceversa.

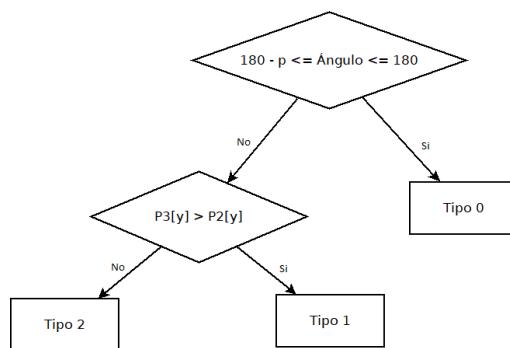


Figura 5.24: Diagrama de flujos penalización comparación versión 2.

Una vez se implementó esta nueva comparación para los brazos, se realizaron una serie de imágenes de prueba para poder confirmar el correcto funcionamiento de la nueva implementación.

## Umbral y penalización

Como se ha comentado, en esta segunda versión, que se centró en el problema con los brazos, se han utilizado dos parámetros: un umbral para definir el rango en el que se considera que un ángulo está estirado o no, y la penalización que se aplica si los tipos son distintos.

Al principio se puso un umbral de 10 grados, que para las imágenes de pruebas que se obtuvieron ofrecieron buenos resultados, diferenciando correctamente cuando una de las partes estaba o no estirada. Lo mejor de que este valor sea un parámetro, es que se puede modificar dependiendo del ejercicio a realizar.

Por otro lado, al implementar la penalización se utilizó el mismo valor tanto para los codos como para los hombros, con una valor de 180 grados de penalización. Pero en comprobaciones que se realizaron después, se observó que penalizar a los hombros con el mismo valor que a los codos no era correcto, ya que la diferencia no era tan grande. Es por ello que se dividió este parámetro de penalización en dos, uno para los codos a 180 y otro para los hombros a 90. Aun así, como pasa con el umbral, estos valores se pueden modificar dependiendo del ejercicio a comparar.

## 5.6. Versión 3 de comparación, uso de medias

Una vez se creía que la comparación de posiciones era correcta había que dar sentido a los valores obtenidos. La forma que se encontró más oportuna para poder realizar esta operación fue el uso de la media de los valores obtenidos en las distintas zonas con las que se trabaja (brazos, piernas y torso).

Además, se implementó el cálculo de la media de tal manera que al aplicar los distintos pesos a las zonas el valor final obtenido siguiese siendo un valor entre 0 y 180 (la distancia máxima que puede haber entre dos ángulos).

Teniendo esto en cuenta, el resultado final se calcula con la siguiente fórmula (siendo *total* la suma de los tres pesos):

$$\begin{aligned} \text{resultado} = & (\text{pesoBrazos}/\text{total}) * \text{compBrazos} + \\ & (\text{pesoPiernas}/\text{total}) * \text{compPiernas} (\text{pesoTorso}/\text{total}) * \text{compTorso} \end{aligned} \quad (5.10)$$

## 5.7. Error en brazos, nueva posición y comparación

En este punto se creía que se tenía una versión final para la comparación de posiciones, pero para probar el funcionamiento de las implementaciones

se realizó una nueva prueba sobre los vídeos originales. En esta segunda prueba se encontró un error muy grave, ya que se penalizaban situaciones en los brazos que nunca se deberían de penalizar.

Estos casos donde se penalizaban eran cuando las manos pasaban de estar por encima de los codos a pasar por debajo, o viceversa. Un ejemplo de esta situación se puede ver en la figura 5.25, donde la diferencia en los brazos entre las dos imágenes se debería calcular únicamente con la diferencia de los ángulos de los codos, pero que con la implementación de la versión 3, además de la diferencia entre los ángulos, se sumaría una penalización de 180 grados a la diferencia.



Figura 5.25: Ejemplo de error en la comparación de brazos.

Ante dicha situación, se plantearon una serie de posibles soluciones a este problema ya reincidente. Primero se intentó dar solución al problema manteniendo el sistema de la posición y el resto de comparaciones que no daban errores, pero no se ideó una solución plausible que solucionara todos los problemas encontrados en las diversas versiones de la comparación de posiciones. Es por ello que se pasó a hacer una cambio más radical, subiendo a un nivel superior el problema, a la clase de *Python* de la posición.

## Nueva versión clase posición

En la primera versión de esta clase, los ángulos se calculaban de forma que nunca eran superiores a 180 grados, lo que daba la necesidad de penalizar ciertas situaciones.

En la segunda versión de la clase se eliminó la condición de que los ángulos fuesen solo de 180 grados (salvo para los ángulos del cuello y de la cadera con el torso, ya que estos ángulos no pueden ser mayores de 180 por fisionomía humana). Para poder expresar el ángulo en un rango de 360 grados se implementó el siguiente algoritmo que utiliza la función anterior de cálculo del ángulo entre 3 puntos:

---

```

#p1,p2,p3: puntos a calcular el angulo ,
# p2 debe ser el punto central
#lado: 0 izquierda , 1 derecha
def calcularAngulo( self ,p1 ,p2 ,p3 ,lado ):
    flag=False
    ang = self .calcularAuxAngulo (p1 ,p2 ,p3 )

    #Cuando la recta que une los dos primeros
    # puntos es una linea vertical
    if (p2[0]-p1[0]) ==0:

        #Lado derecho
        if lado :
            #Si la posicion esta menos a la derecha
            #se ha de cambiar el angulo
            if p3[0]<p2[0]:
                flag=True

        #Lado izquierdo
        else :
            #Si la posicion esta menos
            #a la izquierda se ha de cambiar el angulo
            if p3[0]>p2[0]:
                flag=True

    #Cambio del angulo
    if flag :
        ang = 360 - ang
    else :

```

```

#Calculo de la ecuacion de la recta sustituyendo
# el valor de x por el valor del tercer punto
y = ((p2[1]-p1[1])*(p3[0]-p1[0])/(p2[0]-p1[0]))+p1[1]

#Si esta la tercera parte por encima de la recta que
#hacen las dos primera entonces se cambia el angulo
if p3[1] > y:
    ang = 360 - ang

return ang

```

---

Los pasos que sigue este algoritmo son:

1. Calcular el ángulo en un rango de 180 grados, con la implementación de la anterior versión de la clase de la posición.
2. Comprobar si los dos primeros puntos, de los tres estudiados, forman una línea vertical comprobando la resta en sus valores del eje  $x$ . Si son iguales ir a 3, sino ir a 4.
3. Si los dos puntos forman una línea vertical no tiene sentido comparar la altura con el tercer punto. Es por ello que lo que se hace es comprobar, dependiendo del lado (izquierda o derecho), la posición del tercer punto analizando si este es más cercano al centro del cuerpo (el ángulo se mantiene igual), o si por el contrario está más al exterior del cuerpo donde se realiza una resta de 360 grados menos el ángulo calculado.
4. Si por el contrario los dos puntos no están en una línea vertical (caso más común) se calcula la ecuación de la recta con estos dos primeros puntos. En esta ecuación de la recta, se sustituye el valor de  $x$  por el valor del eje del tercer punto. Por último, se comprueba si el valor de  $y$  calculado con la ecuación de la recta, sustituyendo la  $x$  es superior (se mantiene el ángulo) o inferior al valor  $y$  del tercer punto donde habría que cambiar el valor del ángulo por 360 menos el ángulo calculado al principio del algoritmo.
5. Se devuelve el valor calculado del ángulo.

## Nueva versión comparación

Una vez se había modificado la clase de la posición era necesario hacer una serie de cambios en la comparación de éstas. En este caso, al realizar

estos cambios, ya no hacía falta «discriminar» la comparación de los brazos, ya que ahora todos se calculan de la misma forma y no se necesita ningún umbral ni penalización. Además, aprovechando que no había que realizar ninguna discriminación, por la zona que se estuviese comparando, se realizó una factorización del código implementado para reducir el código repetido.

Para esta cuarta versión de la comparación de las posiciones se realizaron dos funciones. La primera función permite el cálculo de la media de la diferencia entre los ángulos de las zonas pasadas. El código de esta función se puede ver a continuación:

---

```
#pos1, pos2: posiciones a comparar
#zonas: lista de las zonas a comparar
# zonas sin especificar la parte izq o drch
def comparacionZona(pos1, pos2, zonas):
    #Partes derecha e izquierda
    partes = ["D", "I"]
    #Resultado iniciado a 0
    res = 0.0

    #Recorrer las partes
    for i in partes:
        #Recorrer las zonas
        for j in zonas:
            #aux es la diferencia entre los angulos
            aux = abs(eval("pos1." + j + i) -
                      eval("pos2." + j + i))

            #Si la diferencia es mayor de 180
            # grados se coge el otro lado para
            # mantener la distancia minima
            if aux > 180:
                res += (360 - aux)
            else:
                res += aux

    #Se devuelve el valor final entre el
    #numero de elementos sumados
    return res / (len(partes) * len(zonas))
```

---

Como se puede ver en el código, esta implementación permite la comparación de todas las zonas de la posición y siempre devuelve una diferencia

media entre las zonas comparadas que nunca supera los 180 grados, es decir, siempre devuelve la distancia mínima entre los ángulos.

Pero para poder dar distintos pesos a las distintas zonas a comparar, se implementó la segunda función que llama a la función anterior, `comparacionZona`. El código de esta función final que devuelve la diferencia media aplicando los pesos se puede ver a continuación:

---

```
#pos1, pos2 posiciones a comparar
#pesos: diccionario con los pesos de
# cada zona
def compararPosiciones(pos1, pos2, pesos={"brazos":1,
                                             "piernas":1,
                                             "torso":1},
                        res=0,
                        total=0):

    #Se recoge el peso total
    for i in pesos:
        #Si no esta en las keys de la zona se lanza
        # una excepcion
        if i not in zonas:
            raise Exception("Excepcion")
        total+=pesos[i]

    #Se recorren los distintos tipos de zonas
    # y se les aplica el peso a la comparacion
    for i in zonas:
        res+=(pesos[i]/total)*
              comparacionZona(pos1, pos2, zonas[i])

    #Calculo del porcentaje entorno a 180
    porcentaje = res*100/180

return res, 100-porcentaje
```

---

Como se observa en el código, la implementación recorre las distintas zonas de las cuales calcula la comparación entre las posiciones y le aplica el peso considerado. Además, una vez calculada la diferencia final, al estar

está entre 0 y 180 grados por utilizar las medias, se calcula el porcentaje inverso de exactitud entre las dos posiciones. Este porcentaje se interpreta como el porcentaje de exactitud entre las posiciones, siendo el 100 % el valor máximo donde las dos posiciones son totalmente iguales.

## 5.8. Comprobación de la última versión

Una vez se terminaron de implementar estas nuevas versiones, se probaron con las imágenes de la sección 5.5 que la implementación funcionaba correctamente con los datos que las primeras dos versiones tenían grandes problemas. Después, para poder probar el funcionamiento y sobre todo el rendimiento de la implementación, se desarrollaron unas funciones que permiten recorrer todos los vídeos de una carpeta. De cada uno de estos vídeo, la función permite comparar un fotograma con el siguiente de ese mismo vídeo (objetivo principal del proyecto). Cabe comentar en que esta comparación se ha dado el mismo peso a todas las zonas comparadas. Gracias a estas funciones se pudo obtener los siguientes datos de cada uno de los vídeos:

- Media de la diferencia de los grados en todo el vídeo.
- Diferencia máxima en grados en todo el vídeo.
- Diferencia mínima en grados en todo el vídeo.
- Desviación típica entre las diferencias de los grados.
- Media del porcentaje de exactitud.
- Porcentaje máximo de exactitud.
- Porcentaje mínimo de exactitud.
- Desviación típica de los porcentajes.
- Número de fotogramas en el vídeo.
- Tiempo de obtención de las posiciones del vídeo.
- Tiempo de comparación de las posiciones del vídeo.
- Tiempo de cálculo de estadísticas de las comparaciones del vídeo.
- Tiempo total.

## 5.8. COMPROBACIÓN DE LA ÚLTIMA VERSIÓN

49

- Tiempo total empleado por fotograma.

Los resultados obtenidos en esta fase se pueden ver en:

- Resultados diferencia en grados: tabla 5.3.
- Resultados diferencias en porcentajes: tabla 5.4.
- Resultados tiempos de ejecución: tabla 5.5.

Vídeo	MediaGrados	GradosMáximos	GradosMínimos	DesviaciónGrados
depie	5,26	34,89	0,46	5,93
sentado1	2,68	26,07	0,05	3,01
sentado2-cruzado	2,79	15,35	0,40	1,81
sentado2-cruzado-480	2,74	14,59	0,58	1,72
sentado3-caja	2,62	34,32	0,15	4,13
sentado4-remangado	2,78	23,80	0,22	2,96
sentado5-chaqueta-abierta	2,96	17,29	0,42	1,94
sentado6-camiseta	2,36	5,27	0,32	0,96

Tabla 5.3: Tabla con las estadísticas de las diferencias en grados.

Vídeo	PorcMedio	PorcMáximo	PorcMínimo	DesvPorcentaje
depie	97,08	99,74	80,61	3,29
sentado1	98,51	99,97	85,52	1,67
sentado2-cruzado	98,45	99,78	91,47	1,01
sentado2-cruzado-480	98,48	99,68	91,89	0,95
sentado3-caja	98,54	99,92	80,93	2,29
sentado4-remangado	98,46	99,88	86,78	1,65
sentado5-chaqueta-abierta	98,35	99,76	90,39	1,08
sentado6-camiseta	98,69	99,82	97,07	0,53

Tabla 5.4: Tabla con las estadísticas de las diferencias en porcentajes de exactitud.

Vídeo	NFrames	TiempoObtPos (ms)	TiempoComp (ms)	TiempoEst (ms)	TiempoTotal (ms)	Tiempo/Frame (ms)
depie	258	26735,74	50,47	0,21	26786,42	103,82
sentado1	445	50953,61	80,24	0,16	51034,02	114,68
sentado2-cruzado	286	32914,02	44,59	0,13	32958,74	115,24
sentado2-cruzado-480	286	31126,15	44,60	0,12	31170,87	108,99
sentado3-caja	299	33129,63	46,43	0,13	33176,18	110,96
sentado4-remangado	219	24638,31	34,36	0,11	24672,78	112,66
sentado5-chaqueta-abierta	281	31442,28	44,33	0,12	31486,73	112,05
sentado6-camiseta	194	21334,31	31,36	0,11	21365,78	110,13

Tabla 5.5: Tabla con los tiempos de la ejecución.

Como se puede observar, los resultados comparando los fotogramas de un mismo vídeo dan muy buenos resultados, con medias de porcentaje de exactitud que no bajan del 97%, lo que significa que ninguna comparación tiene una media superior a 5,4° de diferencia.

Si se observan los porcentajes máximos de exactitud y los grados mínimos de diferencia, se observan valores muy positivos, siendo los porcentajes muy cercanos al 100% y los grados muy similares a 0°. Este resultado es el que cabría de esperar, ya que en todo el vídeo es muy probable que haya al menos dos fotogramas contiguos que se difieran muy poco al no haber casi movimiento.

Otro dato importante, que principalmente puede servir para detectar fallos en la posición obtenida por la red neuronal, es el porcentaje mínimo de exactitud y el correspondiente grado máximo de diferencia. Observando estos datos se puede ver como los grados máximos de diferencia entre dos fotogramas de todos los vídeos es de unos 35° (que ocurren en el vídeo donde se está de pie). Tras observar en qué fotogramas se haya este máximo se vio como este valor se debía a un pequeño fallo en la detección de la posición por parte de la red neuronal. Pero este «problema» no es importante, debido a que este tipo de movimientos no se realizan en los ejercicios de ejemplos de las rehabilitaciones reales a personas con *Parkinson*, y a que no afecta en gran medida (debido al gran número de fotogramas) a las otras estadísticas, sobre todo a la más importante, la media.

Por último, tras haber comprobado que los algoritmos daban unos resultados bastante positivos, se analizaron los tiempos de ejecución, ya que no había que olvidarse que este proyecto se iba a juntar con el de José Luis Garrido Labrador, para poder realizar estas operaciones en un flujo de imágenes. Es por ello que los algoritmos, además de usar la mínima capacidad de memoria posible, debían de ser rápidos. Este tipo de estudios ya se habían realizado para obtener el mejor modelo de red neuronal que proporcionase los mejores resultados posibles en el menor tiempo posible, por lo que ahora había que comprobar el tiempo de ejecución de todo el proceso, es decir, el cálculo de la posición a partir de la salida de la red neuronal, después la comparación de las posiciones obtenidas y por último el cálculo de estadísticas que permitan obtener un resultado legible y comprensible.

Observando los tiempos de ejecución, tabla 5.5, se puede ver como sin lugar a duda la ejecución del modelo de la red neuronal y el posterior cálculo de la posición es donde se invierte más tiempo (aun teniendo el modelo ya cargado en memoria como se haría en el despliegue final con la estructura de flujos del compañero). Si se mira al resto de tiempos, se puede ver cómo

la implementación de la comparación de posiciones es muy óptima, siendo el tiempo invertido en esta tarea en el vídeo más largo tan solo de 80,24 milisegundos. Por último, el cálculo de estadísticas sobre las comparaciones realizada también es muy rápida, no superando en ninguno de los vídeos los 0,22 milisegundos.

Cabe destacar las diferencias que existen entre los dos vídeos iguales pero con distinta calidad, sentado2-cruzado (720p) y sentado2-cruzado-480 (480p). Como se puede ver en la tabla 5.5, la diferencia entre ambas ejecuciones se difiere en poco más de 1 segundo, que como era de esperar se encuentra en la diferencia en las fases de obtención de la posición a partir de la red neuronal y la clase implementada. Si se observan los resultados de la comparación de los vídeos, en las tablas 5.3 y 5.4, se observa que apenas hay diferencias entre las dos ejecuciones. Teniendo esto en cuenta, se puede asegurar que la reducción de calidad de los vídeos mejora el rendimiento de la herramienta manteniendo unos resultados correctos, al menos si la calidad mínima es 480p.

Como se ha comentado, tras realizar todas la pruebas, se concluyó en que estas versiones del cálculo de posición y de comparación de posiciones serían las implementaciones prácticamente definitivas. Es por ello que interesaba saber el tiempo total invertido para realizar el proceso de obtener la posición de un fotograma, comparar esta posición con la siguiente y obtener las estadísticas de esta comparación. Como se puede ver en la tabla 5.5, el tiempo medio de procesamiento de cada fotograma es de 111,07 milisegundos, un resultado muy bueno si se tiene en cuenta todo el proceso que lleva detrás.

## 5.9. Versión reducida

Tras tener la versión final implementada, se vio necesario la reducción de la clase de la posición para poder ahorrar el mayor tiempo y el mayor uso de memoria posible, objetivo esencial al trabajar con un flujo de datos constante. Una vez se «limpió» la clase para obtener la versión reducida se ejecutaron las pruebas implementadas para la versión anterior para poder observar la mejoría en el tiempo, ya que la salida no se ve afectado (porque solo se eliminaron los cálculos que no se utilizaban en la comparación de posiciones). Los nuevos tiempos de ejecución se puede ver en la tabla 5.6.

Como se puede observar, la diferencia de tiempos apenas se ha visto afectada, reduciéndose solo un poco en el tiempo de obtención de la posición. El nuevo tiempo medio por fotograma es 110,49 milisegundos, lo que significa una mejoría de 0,58 milisegundos. Esta mejoría, al menos a primera vista,

Vídeo	NFrames	TiempoObtPos (ms)	TiempoComp (ms)	TiempoEst (ms)	TiempoTotal (ms)	Tiempo/Frame (ms)
depie	258	26605,28	42,20	0,17	26647,66	104,50
sentado1	445	50694,96	73,0	0,16	50768,12	114,34
sentado2-cruzado	286	32808,08	37,25	0,12	32845,46	116,06
sentado2-cruzado-480	286	30906,14	36,88	0,11	30943,13	109,73
sentado3-caja	299	32520,80	38,96	0,11	32559,87	111,89
sentado4-remangado	219	23845,32	28,89	0,10	23874,31	109,02
sentado5-chaqueta-abierta	281	30808,18	37,17	0,12	30845,47	109,77
sentado6-camiseta	194	21045,52	26,34	0,10	21071,96	108,62

Tabla 5.6: Tabla con los tiempos de la ejecución con la versión reducida de la posición.

no parece gran cosa, pero teniendo en cuenta que se trabaja con flujos, de muchos datos y muy pesados, cualquier mejora es bien recibida. Además, hay que recordar que esta mejoría en el tiempo de ejecución va acompañada de una mejora en la memoria usada, ya que se necesita prácticamente la mitad de memoria para almacenar las variables de la posición comparado con las versiones anteriores.

## 5.10. Preparación para flujos

Tras obtener las versiones definitivas de las implementaciones desarrolladas durante el proyecto, el siguiente paso consistió en pasar todas las implementaciones a un sistema de ficheros *Python* (ya que antes se encontraban en *Jupyter Notebook*). Además, se aprovechó este paso para poder comentar todas las funciones y clases implementadas.

## 5.11. Implementación en el flujo

Esta fue una de las fases más importantes, ya que se trataba de unir los dos trabajos, el realizado por el compañero, José Luis Garrido Labrador, implementando un flujo de vídeos extensible para poder trabajar con él y este trabajo centrado en el análisis y estudio de las posiciones mostradas en los vídeos.

El primer paso que se realizó fue la explicación de cada uno de los trabajos para poder conocer en qué punto se estaba dentro del proyecto. Una vez se conocía el trabajo realizado por cada uno, se pasó a la implementación de la clase `Interfaz` en donde se encuentran todos los métodos necesarios para a partir de una imagen recogida del flujo se pueda:

- Cargar el modelo parametrizando el *threshold* o sensibilidad en la detección de la persona.

- Obtención de la predicción de la posición.
- Cálculo de la posición.
- Comparación de posiciones.

Tras haber implementado esta clase, se pasó a la instalación de la herramienta *Detectron2* en el ordenador personal del compañero donde se encontraba desplegado el flujo para realizar pruebas. Después de haber instalado la herramienta se probó el funcionamiento del flujo con la obtención de la posición, en esta prueba se obtuvo un error debido a que *CUDA* no disponía de la memoria necesaria. Para resolver este problema se implementó una función, que se ejecuta después de calcular la posición de la persona de una imagen, que permite liberar la memoria de *CUDA*. Aún con esta solución, la memoria de la tarjeta gráfica del ordenador personal del compañero no fue suficiente, por lo que se pasó a desplegar el flujo en *Gamma*.

Una vez se realizó la instalación de todas las librerías y herramientas necesarias, se desplegó el flujo. En éste se pudo observar el correcto funcionamiento de ambas partes del proyecto, aunque se encontraron los siguientes problemas:

- Pérdida de fotogramas en el flujo, que se entendieron como fotogramas en donde la herramienta no había conseguido encontrar los puntos necesario o simplemente fotogramas perdidos en el flujo.
- Tiempo de ejecución algo elevado, debido a la necesidad de cargar el modelo de la red neuronal por cada fotograma a predecir. El problema de la carga continua del modelo se debe a cómo está implementado el flujo del compañero, ya que los que realizan las ejecuciones son *workers* que se crean y se destruyen, por lo tanto cada vez que nace un *worker* ha de cargar el modelo de la red neuronal para poder estimar la posición y realizar los distintos cálculos.

Cabe destacar que el tiempo de ejecución en un entorno más realista, es decir, sin tener ni que imprimir ni que almacenar la imagen procesada obtiene una mejora sustancial en el tiempo de ejecución. Es por ello que se implementaron 4 modos de ejecución del flujo:

- **Modo 0:** modo en el que solo se calcula la posición sin necesidad de imprimir ni almacenar el esqueleto. Es el modo más rápido y que más se asemeja al objetivo del proyecto.

- **Modo 1:** modo en el que a parte de devolverse la posición calculada se imprime el esqueleto en la imagen original.
- **Modo 2:** modo en el que a parte de devolverse la posición calculada se imprime el esqueleto en un fondo blanco.
- **Modo 3:** modo en el que a parte de devolverse la posición calculada se imprime el esqueleto en un fondo negro.

## 5.12. Estudio final del flujo

Una vez se había comprobado que la fusión de ambos proyectos era correcta, se realizó un estudio final del flujo paralelizable, para conocer el número de *workers* teóricamente necesarios por cada una de las colas para poder soportar un flujo y una evaluación en tiempo real.

Para ello se realizaron cuatro configuraciones sobre todos los vídeos de pruebas disponibles (un total de 16 vídeos), una en la que se ejecutaba solo la corrección de la imagen y la anonimización de la persona, una segunda donde solo se carga la imagen pero no se realiza ninguna acción (sobre estas dos primeras ejecuciones no se van a comentar nada debido a que es trabajo del compañero), una tercera configuración donde solo se ejecuta el cálculo de la posición de las imágenes y la comparación de las posiciones obtenidas y por último una configuración donde se comprobó la ejecución de todas las partes. Además, de cada una de estas configuraciones se realizó dos ejecuciones, una con el modo 0 donde no se almacenaba la imagen, y otra en modo 1 donde se guardaba la imagen procesada.

Para poder obtener unos resultados más realistas, ya que tras varias ejecuciones el tiempo usado variaba considerablemente (principalmente porque la estación de trabajo no solo se usaba para este trabajo, sino que había más investigadores de la universidad usándola), se ejecutó todo el proceso 10 veces, siendo los datos de resultado el tiempo medio de las 10 ejecuciones que el sistema había tardado en procesar.

Para calcular el número de *workers* mínimos necesarios para ejecutar el flujo en tiempo real se usó la siguiente fórmula:

$$workers = (MediaTiempoPorFrame + Desviacion * 2) * fps / 1000 \quad (5.11)$$

En la fórmula se puede ver como se usa la media de tiempo por fotograma (en milisegundos) más 2 veces la desviación típica del tiempo por fotograma

para poder tener así en cuenta el 95 % de los casos (aproximando la distribución obtenida a una distribución normal, la justificación se encuentra en las gráficas de la distribución de cada ejemplo). El número de fotogramas por segundo o *fps* se ha calculado con dos valores muy representativos: 5 *fps* y 15 *fps*.

## Resultados solo posición y comparación

Los resultados obtenidos en esta ejecución se pueden ver en la tabla 5.7, donde la fila «Guardando» representa la ejecución con el modo 1, y la fila «Sin guardar» representa la ejecución con modo 0.

Tipo	Fotogramas	Media	Desv	Min	25 %	50 %	75 %	Max
Guardando	4863	463,00	117,58	133,33	376,03	455,41	543,79	841,62
Sin guardar	4863	313,50	94,25	67,19	243,11	311,38	379,01	618,90

Tabla 5.7: Tabla con los resultados de la ejecución solo de la posición y comparación del flujo en milisegundos.

Para poder observar mejor se realizó una gráfica, figura 5.26, donde se puede comprobar como en la ejecución sin guardar la imagen la media se centra en unos 313 milisegundos, y de 460 milisegundos en el modo 1 con guardado de la imagen. Estos datos difieren un poco de los obtenidos en el resto de pruebas realizadas, es por ello que se entienden como un retardo debido a la estructura simulada del flujo que se realizó en el entorno de prueba (flujo del compañero pero sin contar con *Kafka*). Aun así, los resultados obtenidos son muy positivos, ya que por ejemplo en el modo 0 nunca se superan los 620 milisegundos por fotograma, y el número de *workers* necesarios es bajo como se verá a continuación.

Como se puede observar en la figura 5.26, existen algunos fotogramas que obtienen tiempos elevados en ambas ejecuciones. La existencia de estas situaciones se tuvieron que tener en cuenta en el cálculo del número de *workers* necesarios en la paralelización del flujo. Debido a que en una situación ideal no habría apenas variación entre los tiempos de ejecución lo permitiría acortar de una manera más justa este número.

El número de *workers* necesarios para ejecutar el flujo sin almacenar el procesado del esqueleto (modo 0) realizando solo las tareas de cálculo de

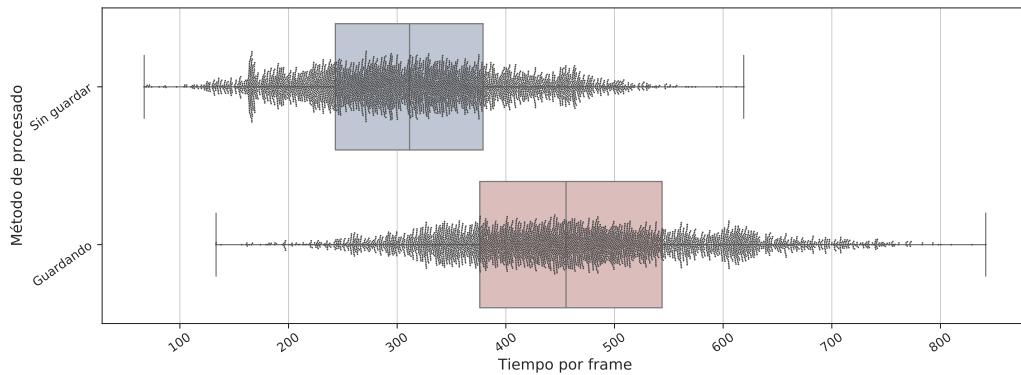


Figura 5.26: Resultado ejecución de solo la posición y comparación del flujo en milisegundos.

posición y comparación de posiciones es:

$$\begin{aligned} workers_{5FPS} &= (313,50 + 94,25 * 2) * 5 / 1000 = 2,51 \\ workers_{15FPS} &= (313,50 + 94,25 * 2) * 15 / 1000 = 7,53 \end{aligned} \quad (5.12)$$

Este resultado daría que como mínimo se deberían de utilizar 3 (5 *fps*) y 8 (15 *fps*) *workers* a los que para que paliar los casos especiales, donde la duración del procesamiento es excesiva, se deberían de añadir al menos uno o dos.

Por otro lado, el número de *workers* necesarios en el caso de que se quiera almacenar la imagen es:

$$\begin{aligned} workers_{5FPS} &= (463,00 + 117,58 * 2) * 5 / 1000 = 3,49 \\ workers_{15FPS} &= (463,00 + 117,58 * 2) * 15 / 1000 = 10,47 \end{aligned} \quad (5.13)$$

Este resultado muestra la necesidad de tener un total de 4 (5 *fps*) y 11 (15 *fps*) *workers*, a los que como en el caso anterior habría que sumar uno o dos más para las situaciones extraordinarias.

Para justificar la aproximación a la distribución normal para el uso de la media más dos veces la desviación ( $2\sigma$ ), se puede observar la distribución de los tiempos en la figura 5.27. En la imagen se puede observar como cogiendo estos valores para el cálculo del número de *workers* se está teniendo en cuenta el tiempo de prácticamente todas las ejecuciones.

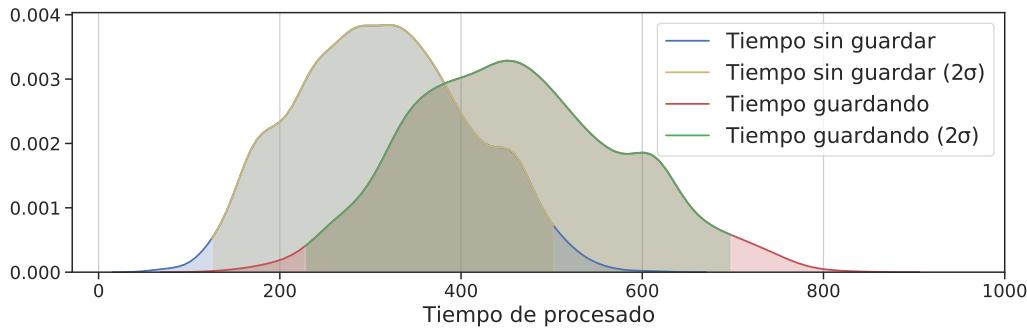


Figura 5.27: Distribución tiempo medio de ejecución solo posición y comparación.

## Resultados flujo entero

Los resultados obtenidos de la ejecución del flujo con todos sus componentes se puede ver en la tabla 5.8.

Tipo	Fotogramas	Media	Desv	Min	25 %	50 %	75 %	Max
Guardando	4863	626,08	98,76	219,33	564,79	636,96	692,34	932,61
Sin guardar	4863	464,04	86,52	193,62	410,99	468,62	521,74	702,43

Tabla 5.8: Tabla con los resultados de la ejecución del flujo en milisegundos.

Como se puede observar, los tiempos medios de procesamiento de cada fotograma han aumentado unos 150 milisegundos en el modo sin guardar y unos 200 milisegundos en el modo con guardado añadiendo el resto del flujo. Para poder observar mejor los resultado obtenidos se realizó la gráfica que se puede ver en la figura 5.28. En esta gráfica se puede ver como el tiempo medio de ejecución es de 460 milisegundos en el modo sin guardado que es un resultado muy bueno, que como se ve a continuación necesita un número de *workers* paralelizados muy bajo.

Cabe destacar, que en el modo en el que se debería desplegar el sistema, el modo 0 sin almacenar ninguna imagen, se obtienen resultados buenísimos, siendo el máximo tiempo invertido por fotograma de poco más de 700 milisegundos.

El cálculo de los *workers* necesarios para hacer el flujo paralelizable en tiempo real en el modo sin guardado se puede ver a continuación:

$$\begin{aligned} workers_{5FPS} &= (464,04 + 86,52 * 2) * 5 / 1000 = 3,19 \\ workers_{15FPS} &= (464,04 + 86,52 * 2) * 15 / 1000 = 9,56 \end{aligned} \quad (5.14)$$

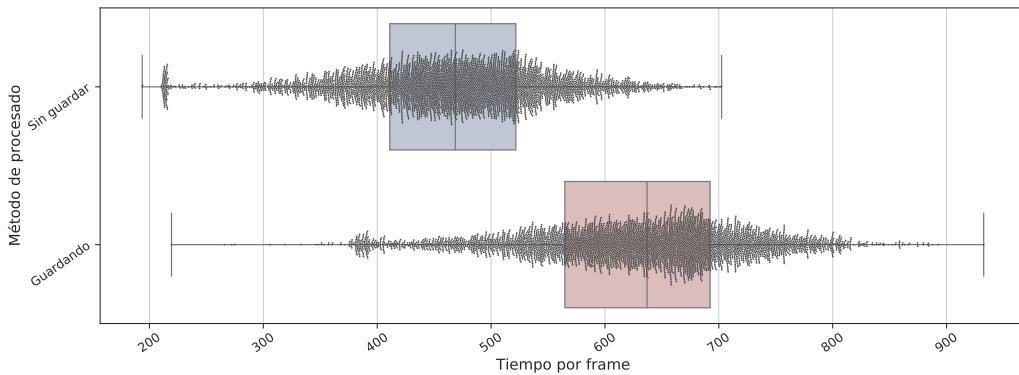


Figura 5.28: Resultado ejecución flujo en milisegundos.

Con estos resultados se necesitan un total de 4 *workers* en el caso de trabajar con 5 fotogramas por segundo, y de 10 *workers*, como mínimo, para trabajar con vídeos de 15 fotogramas por segundos. Como en el estudio anterior, es más que recomendable añadir uno o dos *workers* más a este valor para poder sobre llevar los casos extraordinarios.

Para el modo en el que se procesa y almacena la imagen se necesitan los siguientes números de *workers*:

$$\begin{aligned} workers_{5FPS} &= (626,08 + 98,76 * 2) * 5 / 1000 = 4,11 \\ workers_{15FPS} &= (626,08 + 98,76 * 2) * 15 / 1000 = 12,35 \end{aligned} \quad (5.15)$$

A partir de este cálculo se puede afirmar que para poder procesar un vídeo en tiempo real y obtener las imágenes resultantes se deberían de usar al menos 5 *workers* en el caso de que el vídeo se retransmita a 5 fotogramas por segundos, y de 13 *workers* si el vídeo es de 15 fotogramas por segundos. Como en todos los casos anteriores, se recomienda añadir al menos un *worker* más para asegurar el correcto funcionamiento ante ejecuciones extraordinarias.

Como en el caso anterior, se muestra la distribución de los resultados (figuraX 5.29) para comprobar que con la aproximación a una distribución normal y el uso de dos veces la desviación se recogen la mayoría de los resultados.

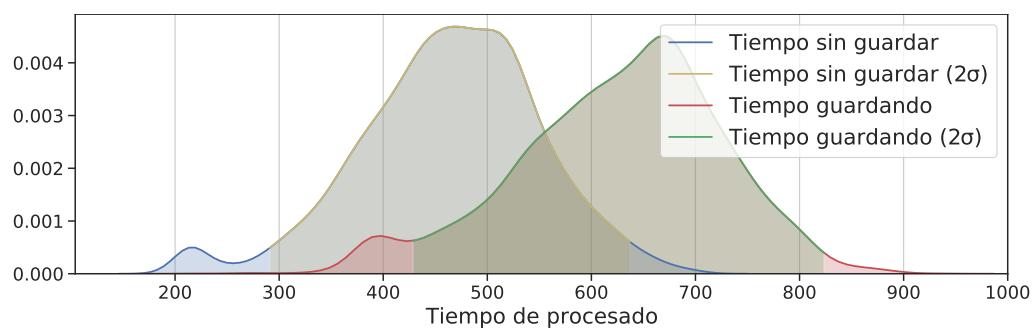


Figura 5.29: Distribución tiempo medio de ejecución flujo.



---

## Trabajos relacionados

---

La rehabilitación de pacientes con problemas físicos o mentales ha evolucionado en los últimos años, gracias a los avances tecnológicos, a un sistema de rehabilitación *online*, que como se explico en la introducción del trabajo, permite habilitar estos sistemas de rehabilitación a pacientes que por diversas razones no pueden desplazarse a las consultas médicas a realizar esta rehabilitación para mejorar sus situación.

La aparición de las rehabilitaciones *online* están permitiendo la automatización de estos sistemas. Gracias a herramientas de visión por computador permiten, sin la presencia de un médico especializado, dar una retroalimentación al paciente de los ejercicios que realiza.

A continuación, se van a comentar algunos de los proyectos, que como el proyecto que se ha desarrollado en este Trabajo Fin de Máster, han desarrollado un sistema que utiliza la visión por computador para mejorar las rehabilitaciones de pacientes.

### **A Kinect-based system for cognitive rehabilitation exercises monitoring [16]**

En este *paper* de la Universidad de Valladolid se puede observar el uso de técnicas de visión por computador para calcular características como el tiempo de reacción de los pacientes con algún problema físico o personas que confunden el lado derecho e izquierdo de su cuerpo, y seguir los movimientos del paciente.

Los ejercicios en los que se centra este proyecto se centran en movimientos de la mano hacia la cabeza, para ello utilizan la estimación de posición, en 3D, que les ofrece la cámara *Kinect*. *Kinect* es una cámara desarrollada

por *Microsoft* que originalmente se diseñó desde un enfoque lúdico para la vídeo consola *Xbox 360*, pero que ha terminado siendo una de las mejores cámaras para su uso en visión por computador [29]. Actualmente, con la nueva generación de la vídeo consola de *Microsoft* (*Xbox One*) salió en 2013 una nueva versión de su cámara *Kinect*.

Para poder monitorizar estos ejercicios, se realiza una estimación y seguimiento de los rasgos faciales (ojos, nariz y orejas) y de las manos derecha e izquierda. Para realizar este proceso se utiliza la estimación de la posición dada por *Kinect* y el uso de modelos predictivos como *AdaBoost* [14] para la detección de la cara, de la cual posteriormente se los rasgos faciales (el esqueleto proporcionado por *Kinect* solo aporta la posición de la cabeza, no de los rasgos faciales).

En este proyecto se trabajó con vídeos de  $640 \times 480$  píxeles con 13 fotogramas por segundo, con un tiempo medio de estimación de las posiciones necesarias de 79 milisegundos, realizado en Visual C++ en un dispositivo con un procesador de 3 GHz. La evaluación de los ejercicios se realiza de manera *offline* una vez termina el ejercicio.

Cabe destacar que este proyecto se desarrolló con la finalidad de ser incluido en *GRADIOR*<sup>3</sup>, plataforma virtual para la estimulación cognitiva, evaluación y rehabilitación neuropsicológica [18].

## Computer Vision-Based Classification of Hand Grip Variations in Neurorehabilitation [33]

En este proyecto se realizó un sistema de visión por computador, que puede usarse con realidad virtual, para la clasificación de posturas de las manos en la rehabilitación de pacientes.

Para poder obtener mayor información para detectar correctamente la postura de la mano se utilizan dos cámaras, una que graba el lateral de la mano, y otra que graba la parte superior de ésta (figura 6.30). Las cámaras capturan con una frecuencia de 15 fotogramas por segundo a una resolución de  $352 \times 288$  píxeles.

Para el procesado de las imágenes se usa una librería de C++ llamada OpenCV. Con esta librería se quita el fondo de la imagen para dejar solo a la mano (con técnicas de comparación de colores, erosión y dilatación de la imagen). Sobre esta imagen procesada se obtiene:

---

<sup>3</sup>GRADIOR: <https://ides.es/gradior>

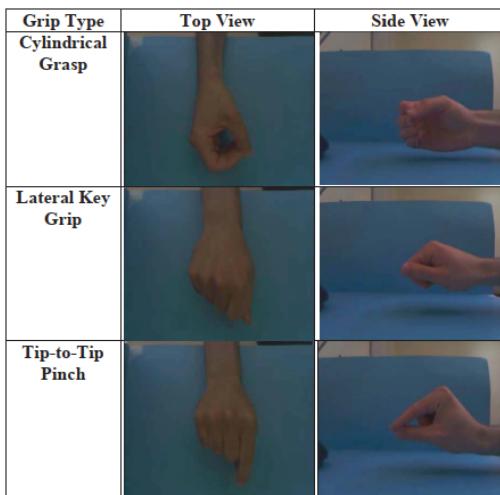


Figura 6.30: Ejemplos cámaras en rehabilitación de manos [33].

- Los siete *Hu invariant moments* sobre una versión de escala de grises de la imagen [17].
- Contorno de la mano.
- Contorno interior en los casos necesarios, cuando la mano hace la forma de un agujero.

La clasificación final de las posturas se realiza con estos datos utilizando *k-Nearest Neighbors* (*kNN*), calculado con los 20 vecinos más cercanos ( $k = 20$ ).

## A Computer Vision System for Virtual Rehabilitation [3]

En este proyecto se utiliza la segunda versión de la cámara de *Microsoft Kinect*, para la mejora en la estimación de la postura y el seguimiento del movimiento en personas con deficiencias motoras. Esta mejora se consigue utilizando las distintas cámaras de *Kinect V2* y juntando toda esa información para obtener el mejor esqueleto posible en 3D. Además, el proyecto ha sido desarrollado para ser usado con realidad virtual con *Oculus Rift 2 HMD*.

Este proyecto destaca por el uso de 4 *Kinect V2* puestas en las esquinas de la habitación conectadas a un *Intel Next Unit of Computing (NUC)* que

permite preprocesar las imágenes, solo unos pocos puntos son mandados al ordenador maestro (*Master PC*) para crear el esqueleto (figura 6.31).

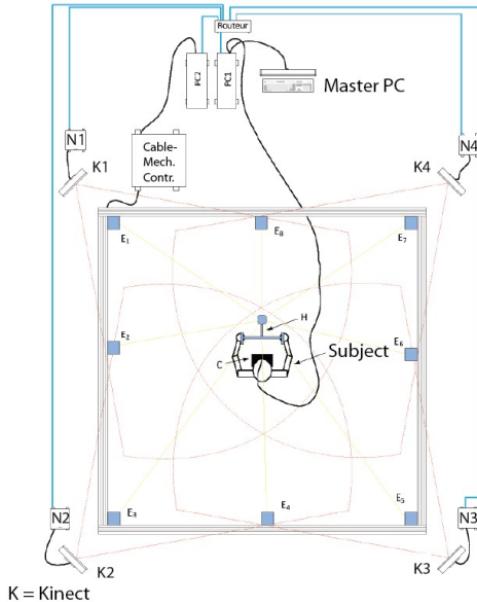


Figura 6.31: Disposición *Kinects V2(K)*, *NUC(N)* y el *Master PC* [3].

En el *paper* los creadores comentan que la estimación de la posición de *Microsoft* es muy buena, pero que al tener 4 *Kinects V2* se tiene que juntar y aplicar unos filtros para el correcto funcionamiento.

Además, en el *paper* se describen dos formas de ejecutar la estimación de la posición desarrollada, una versión en «tiempo real» que permite mandar la información del esqueleto directamente a las *Oculus*, y una versión *offline* de mayor precisión pero con la necesidad de un postprocesado de la información obtenida.

## Conclusiones

Todos estos proyectos, cada uno orientado y elaborado de una forma distinta, muestran lo valioso que puede llegar a ser la introducción de la tecnología en la rehabilitación de pacientes con problemas físicos e incluso mentales. Este valor añadido ofrece, además de una reducción en costes y en intervención humana, una mejoría en la vida de los pacientes ya que realizan más rehabilitaciones que hacen sus vidas más llevaderas [4].

---

# **Conclusiones y Líneas de trabajo futuras**

---

En este apartado se encuentran las conclusiones finales del proyecto, así como el conjunto de tareas y/o mejoras que se podrían realizar.

## **7.1. Conclusiones**

El proyecto se ha desarrollado en una situación anómala. Con la irrupción del *COVID-19* y el correspondiente confinamiento se limitó mucho el desarrollo orientado al paciente que se estaba dando en las primeras fases del desarrollo de la aplicación de rehabilitación *online*. Con suerte, para mediados de febrero ya existía una primera versión de esta aplicación y se pudo realizar, antes de cualquier indicio del virus en el país, la primera instalación de los dispositivos en casa del primer paciente (aunque por problemas con la red *WiFi* se tuvo que quitar).

La imposibilidad de realizar instalaciones de dispositivos impidió la recolección de datos (vídeos) con los que poder implementar un modelo propio de visión artificial. Aunque al realizar el estudio del estado del arte sobre las herramientas existentes se vio que hubiese sido muy complicado la creación de uno de estos modelos principalmente por limitaciones en el *hardware*. Aun así, el estudio del estado del arte realizado ha dado muy buenos resultados, encontrando una herramienta muy buena que permite realizar las tareas de visión por computador en unos tiempos muy buenos.

A partir del estudio de las herramientas y con la obtención del modelo final se pudo estudiar e implementar un sistema de extracción de características de la salida del modelo, con el que posteriormente se desarrolló

la comparación de posiciones. Aun con la dificultad de no poder realizar reuniones ni charlas presenciales para comentar el proceso del desarrollo con el equipo médico, sí que se realizaron reuniones de manera *online* y se recibieron vídeos de ejercicios de ejemplos en los que se basaron principalmente para realizar la comparación de posiciones.

Con todas estas tareas realizadas se cumplió con todos los objetivos marcados al principio del proyecto, incluso superando con creces algunas expectativas sobre el funcionamiento del estimador de posiciones y de los tiempos de ejecución en un problema tan complejo y con una gran cantidad de datos muy pesados.

Actualmente, con la vuelta a la normalidad se está volviendo a realizar instalaciones de dispositivos con los cuales se están recogiendo los primeros datos. Gracias al proyecto, los pacientes con *Parkinson* no necesitarán desplazarse para realizar sus rehabilitaciones y podrán realizarlas de una manera más frecuente con el sistema de retroalimentación de sus ejercicios y evoluciones. Además, con la implementación extensible realizada con la configuración de las comparaciones estas implementaciones se puede utilizar para distintos tipos de ejercicios y de enfermedades.

Por último, cabe destacar el interés por parte de la OTRI-Transferencia (servicio de la Universidad de Burgos centrado en el desarrollo de ideas y del fomento del emprendimiento en los estudiantes) al recibir el premio prototipo de la edición 2019-2020<sup>4</sup>. Gracias a este premio se pudo asistir a distintos talleres de comunicación y presentación de proyectos, marketing digital y de propiedad industrial. Además, se recibió un asesoramiento sobre estos ámbitos.

## 7.2. Líneas de trabajo futuras

Como se ha comentado, la principal línea de trabajo futura, que ya se está trabajando pero que no entra en este Trabajo Fin de Máster, es la incorporación del proyecto del compañero José Luis Garrido Labrador y este proyecto a la aplicación *web*. Para así permitir a los pacientes realizar ejercicios de forma autónoma.

Aun así, las principales líneas futuras de trabajo orientadas al desarrollo único de este proyecto son:

<sup>4</sup>Resolución Premios Prototipo 2019-2020: <https://www.ubu.es/te-interesa/convocatoria-prototipos-orientados-al-mercado-curso-2019-2020>.

- Poder probar a crear un modelo propio de visión artificial. Por insuficiencia de datos esta meta no se pudo si quiera plantear.
- Contar con un mayor presupuesto que permita obtener cámaras de mayor calidad como pueden ser las *Kinect V2* que permiten la obtención de la posición en 3D y mejores ordenadores en las casas de los pacientes que permitan realizar un preprocesado o incluso la extracción de la posición. Convirtiendo así el flujo de vídeos en un flujo de posiciones con un tamaño considerablemente menor, lo que permitiría mejorar los tiempos del flujo aun más.
- Extraer mayor información más allá de la comparación de ejercicios, como puede ser el tiempo de reacción de los pacientes.
- Implementar en paralelo un predictor de caídas. Los pacientes de *Parkinson* suelen sufrir de caídas debido a las limitaciones en su sistema motor. Por ello un sistema capaz de predecir la caída del paciente para poder detener el ejercicio o incluso alertar a algún familiar o conocido.



# Apéndices



## *Apéndice A*

---

# **Plan de Proyecto Software**

---

## **A.1. Introducción**

La planificación de un proyecto es una fase esencial en desarrollo de éste, ya que permite comprobar y guiar el proyecto según las necesidades actuales y futuras.

Dentro de este plan para el desarrollo del proyecto se pueden diferenciar dos puntos sobre los que se puede enfocar este estudio:

- **Temporal:** enfoque en el que se analiza la evolución del proyecto en el tiempo. Al utilizar una metodología SCRUM, se ha dividido el desarrollo en *sprints*, que dependiendo de las funcionalidades a desarrollar, y debido a la situación que se ha pasado con el COVID-19, han tenido distintas duraciones.
- **Viabilidad:** comprobación de la adecuación del proyecto tanto desde el ámbito económico como desde el ámbito legal.

## **A.2. Planificación temporal**

Como ya se ha comentado, la planificación temporal se ha dividido en distintos *sprints* de 2 semanas cada uno, en los cuales se realizaron distintas tareas y reuniones. Cabe destacar que antes de realizar el primer *sprint* se desarrolló la aplicación FIS sobre la cual será implementado en el futuro el proyecto desarrollado.

## Sprint 0

En este *sprint* se creó la aplicación para el Hospital Universitario de Burgos llamada FIS-HUBU. Esta aplicación, como se ha visto en el apartado 5.1, sirve para realizar vídeo llamadas entre médicos y los pacientes con Parkinson en las cuales se realizan rehabilitaciones para mejorar su calidad de vida. Como se ha explicado, esta aplicación cuenta con dos partes, una para los médicos o responsables y otra para los pacientes. Las tareas realizadas en este *sprint* son las siguientes:

- Investigación sobre plataformas de vídeo llamadas.
- Creación de la estructura base de la página web.
- Diseño e implementación de la base de datos.
- Creación del login por usuario, guardando una *cookie* para mantener la sesión.
- *Backend* de las páginas de los pacientes, donde se incluye la creación de las llamadas, el calculo de la evolución a partir de los datos en la base de datos, la grabación de la cámara del paciente y subida a un servidor propio de la universidad...
- *Backend* de las páginas de los responsables.

Tras desarrollar la aplicación se empezaron con las tareas de explotación de la aplicación, en donde se cogieron los dispositivos para los paciente y se les instalaron todo el *software* necesario para funcionar correctamente, y con ello poder grabar los vídeos y subirlos a un servidor propio. Debido a la aparición del COVID-19 y el posterior confinamiento sufrido y las consiguientes medidas de seguridad no se pudo empezar a instalar los dispositivos en las casa de los pacientes.

## Sprint 1: 17/02/2020 - 26/02/2020

Este es el primer *sprint* real del desarrollo de la parte del proyecto de visión artificial, en el se realizaron las primeras tareas de creación del repositorio en *Github* y la creación de la estructura del mismo. Además, se creo el documento *LATEX* a partir de la plantilla de la asignatura. Las tareas más relevantes en este primer *sprint* fueron las relacionadas con la configuración de *Gamma* y los primeros pasos en la investigación de algoritmos de visión por computador y detección de movimiento, como son *Detecron2* y *PoseNet*.

**Sprint 2: 27/02/2020 - 11/03/2020**

En este *sprint* se realizó una tarea fundamental dentro del desarrollo del proyecto, que es la investigación y elección del algoritmo de visión artificial para la obtención de la postura, que se usará en los fotogramas de los vídeos recogidos con la aplicación FIS-HUBU y para compararlos con otros vídeos base para saber la exactitud del ejercicio. Esta tarea de investigación de los distintos algoritmos dio como resultado que el *Detectron2* (apartado 4.1) es el algoritmo que más se amoldaba al problema del proyecto.

Además, en este *sprint* se realizaron las tareas de documentación necesarias sobre los algoritmos candidatos y sobre la selección del mejor.

**Sprint 3: 12/03/2020 - 02/05/2020**

En este tercer *sprint* se profundizó en la investigación de los distintos algoritmos y modelos de *Detectron2*. En este *sprint* se realizaron las siguientes tareas:

- Documentación de los primeros objetivos del proyecto.
- Estudio de la herramienta *Detectron2*.
- Estudio de los modelos ya creados en *Detectron2*.
- Estudio sobre los modelos de detección de posición de *Detectron2*.
- Estudio del efecto del *threshold* en la detección.
- Interpretación de la salida obtenida.
- Creación de la clase de posición y extracción de características a partir de los datos devueltos por el modelo.
- Documentación de los aspectos relevantes.

**Sprint 4: 03/05/2020 - 05/06/2020**

Este *sprint* se realizaron tareas muy importantes tanto en la documentación como en la implementación, ya que se documentó el concepto teórico más importante, las redes neuronales convolucionales, y se implementó las primeras versiones de la comparación de posiciones. Las principales tareas que se realizaron en este *sprint* son:

- Documentación de los conceptos teóricos, donde destaca la documentación teórica de las redes neuronales convolucionales que tiene el mayor peso teórico del proyecto.
- Documentación de las técnicas y herramientas usadas hasta el momento.
- Cambios en la memoria de la primera revisión por parte de los tutores.
- Desarrollo de la primera versión de comparación de posiciones.
- Implementación de la corrección del *bug* en la comparación de los brazos.
- Creación de imágenes para probar la comparación de posiciones.
- Test con las imágenes obtenidas.
- Documentación de los aspectos relevantes ocurridos en este *sprint*.

### ***Sprint 5: 06/06/2020 - 12/06/2020***

En este *sprint* se ha terminado de implementar las versiones finales de la posición y de la comparación de estas, tras volver a encontrar un error en las pruebas. El conjunto de tareas que se han realizado en este *sprint* son:

- Creación de la versión 3 de la comparación de posiciones con la introducción del cálculo de la media.
- Corrección del error en la comparación de los codos, nueva versión de la clase de la posición.
- Implementación sistema de comprobación y extracción de estadísticas.
- Creación de versión reducida de la clase de la posición y pruebas temporales.
- Paso de todas las versiones implementadas a ficheros *Python*.
- Documentación de los aspectos relevantes del *sprint*.
- Obtención de la imagen solo con el esqueleto calculado.
- Modificaciones hechas por la segunda revisión de la memoria.

**Sprint 6: 13/06/2020 - 27/07/2020**

Este fue el primer *sprint* de la recta final del proyecto, donde se comenzó con la unión del proyecto con el flujo implementado del compañero y con las pruebas referentes a este proceso, que por la cantidad de datos y pruebas que se realizaron no se pudo terminar en este *sprint*. Además, en este *sprint* se comenzó a documentar las distintas partes de la memoria que se tenían planteadas pero no documentadas, en este apartado entra:

- Documentación del manual del programador.
- Documentación del diseño.
- Documentación de los trabajos relacionados.
- Documentación de la introducción del documento.
- Documentación del resumen y *abstract*.
- Documentación de más aspectos teóricos.
- Documentación de las conclusiones y líneas futuras de trabajo.

## A.3. Estudio de viabilidad

En este apartado se va a comentar la viabilidad del proyecto tanto de forma económica calculando el coste total que debería de haber costado el desarrollo del proyecto, y la viabilidad legal de las librerías y herramientas utilizadas.

### Viabilidad económica

### Viabilidad legal

En este subapartado se va a comentar las distintas licencias que tienen las herramientas y librerías utilizadas en el proyecto, así como se comenta la licencia final que tiene el proyecto.

Las licencias de las librerías y herramientas utilizadas en el desarrollo del proyecto se pueden ver en la tabla A.1.

Librería-Herramienta	Licencia
<i>Numpy</i>	BSD 3
<i>Pickle</i>	PSF
<i>Pandas</i>	BSD 3
<i>OpenCV</i>	BSD 3
<i>Garbage Collector</i>	PSF
<i>Pynvml</i>	BSD
<i>Matplotlib</i>	PSF
<i>Seaborn</i>	BSD 3
<i>Plotly</i>	MIT
<i>Pytorch</i>	BSD
<i>Torchvision</i>	BSD 3
<i>Math</i>	PSF
<i>Os</i>	PSF
<i>Detectron2</i>	Apache 2.0

Tabla A.1: Tabla con las licencias de las librerías y herramientas utilizadas.

## *Apéndice B*

---

# Especificación de diseño

---

## B.1. Introducción

En cualquier proyecto la fase de diseño de las distintas partes de éste es esencial para su correcta implementación, desarrollo y evolución. En este apartado se va a comentar los distintos diseños que se han realizado para poder obtener una solución óptima, aun así hay que tener en cuenta el gran peso que tiene la investigación en este proyecto por lo que el diseño en una gran parte del proyecto no ha sido necesario.

Los diseños realizados en el proyecto han sido:

- **Diseño de datos:** subapartado donde se van a mostrar las distintas estructuras de datos utilizadas, así como un conjunto de diagramas para comprender la estructura de los ficheros de ejecución del proyecto.
- **Diseño procedimental:** subapartado donde se va a mostrar principalmente como es la comunicación entre el flujo y las implementaciones realizadas.

## B.2. Diseño de datos

Este proyecto al ser mayoritariamente un proyecto de investigación apenas tiene diseño de datos. Aun así , sí que existe un diseño donde se muestra la estructura de los ficheros necesarios para la ejecución final sobre el flujo.

Este diseño se muestra en forma de diagrama de clases, figura B.1, donde los asteriscos en los nombres de las variables representan que realmente hay dos variables una para la parte izquierda y otra para la derecha (un ejemplo puede ser el hombro\*, que significa que existe un hombroI y un hombroD, los dos del mismo tipo). En el diagrama se muestra la implementación final de la posición con la clase Posicion, que como se comentó en el apartado 5.9, es la versión reducida con el conjunto mínimo de cálculos necesarios.

Por otro lado, la clase Interfaz es la que se comunica con el flujo de datos (de ahí su nombre), en ella se carga el modelo y se crean y se comparan las posiciones.

Posicion	Interfaz
<pre>+nariz: [float,float] +hombro*: [float,float] +cuello: [float,float] +angCuelloSup*: float +codo*: [float,float] +mano*: [float,float] +angCodo*: float +angHombro*: float +cadera*: [float,float] +cadera: [float,float] +rodilla*: [float,float] +angCadera*: float +angCaderaTorso*: float +tobillo*: [float,float] +angRodilla*: float +__init__(x:list(float),y:list(float)) +calcularPuntoMedio(p1:[float,float],p2:[float,  float]): [float,float] +calcularAuxAngulo(p1:[float,float],p2:[float,  float],p3:[float,float]): float +calcularVector(p1:[float,float],p2:[float,  float]): [float,float] +calcularAngulo(p1:[float,float],p2:[float,  float],p3:[float,float],lado:boolean): float</pre>	<pre>+cfg: cfg +predictor: DefaultPredictor +__init__(threshold:float=0.99) +_borrarMemoria() +obtenerPosicion(imagen:[int],codigo:int=0): Posicion, [int] +&lt;&lt;static&gt;&gt; compararPosiciones(pos1:Posicion,                                      pos2:Posicion,                                      pesos:dict={"brazos":1,  "piernas":1,  "torso":1}): float, float +&lt;&lt;static&gt;&gt; comparacionZona(pos1:Posicion,                                     pos2:Posicion,                                     zonas:[string]): float</pre>

Figura B.1: Diagrama de clases.

### B.3. Diseño procedimental

El diseño procedimental realizado en este proyecto, como en el caso del diseño de datos, se ha realizado sobre la implementación final en las clases que usan el flujo de datos. Estos diseño procedimentales se han realizado a partir de diagramas de secuencia, donde se muestran las dos posibles interacciones del flujo con las clases.

En la primera posibilidad, el flujo tan solo pide las posiciones de las imágenes que va proporcionando, figura B.2. Como se puede ver en el diagrama de secuencias, los primero que tiene que realizar un *worker* del flujo es instanciar la clase Interfaz donde se procede con la carga y configuración del modelo. Una vez instanciada la clase se puede hacer llamadas para obtener

la estimación de la posición de un imagen (operación que se puede repetir mientras esté cargado el modelo, es decir, mientras esté vivo el *worker*).

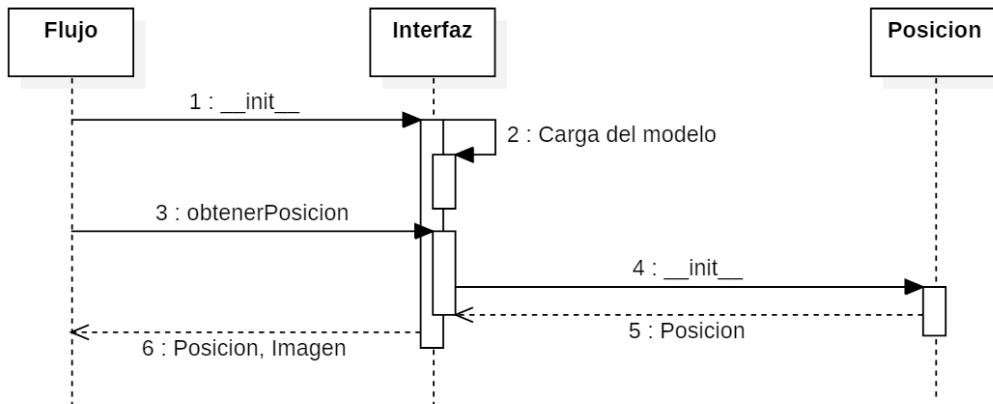


Figura B.2: Diagrama de secuencia, estimación posición.

Por otro lado, una segunda posibilidad de la ejecución con el flujo es la comparación de posiciones, figura B.3. En este diagrama de secuencias se muestra como es la interacción entre los distintos elementos que trabajan con el flujo, como en el caso anterior las llamadas a la comparación y obtención de posiciones se pueden realizar mientras esté vivo el *worker* que tiene el modelo cargado.

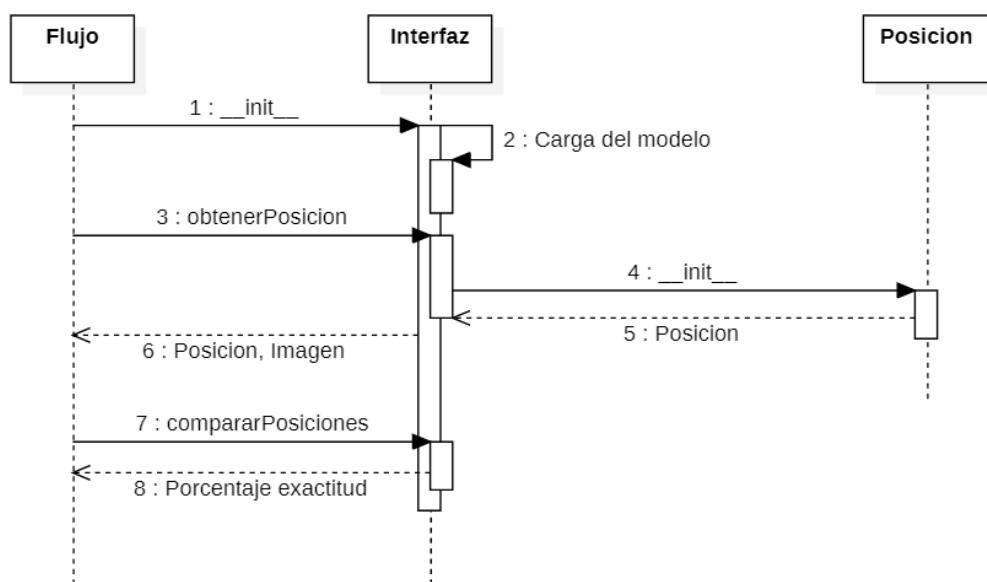


Figura B.3: Diagrama de secuencia, comparación de posiciones.

## *Apéndice C*

---

# **Documentación técnica de programación**

---

## **C.1. Introducción**

En este apartado se encuentran los puntos que ha de conocer una persona, con conocimientos en informática, que quiera utilizar los distintos elementos del proyecto. Es por ello que este apartado se divide en los siguientes subapartados:

- **Estructura de directorios:** subapartado donde se comenta la estructura del proyecto. El proyecto además se puede observar en el repositorio de *GitHub*, donde a parte de la estructura y los documentos del proyecto se pueden observar las tareas que se han realizado.
- **Manual del programador:** manual en donde se muestran los pasos a seguir para poder continuar desarrollando el proyecto.
- **Compilación, instalación y ejecución del proyecto:** explicación paso a paso para conseguir utilizar los distintos elementos implementados en el proyectos.
- **Pruebas del sistema:** explicación de las distintas pruebas realizadas en la fase de investigación y de despliegue del proyecto, así como explicación de como lanzar estas pruebas.

## APÉNDICE C. DOCUMENTACIÓN TÉCNICA DE PROGRAMACIÓN

### C.2. Estructura de directorios

En este subapartado se va a comentar la estructura y el contenido de los directorios del proyecto que se encuentra subido a un repositorio público en *GitHub*<sup>1</sup>. Empezando con la raíz del proyecto que tiene la siguiente estructura:

```
/  
└── src  
└── doc  
    ├── README.md  
    ├── LICENSE  
    └── .gitignore
```

Como se puede ver, de la raíz cuelgan dos carpetas, src y doc, y tres documentos que se usan para el repositorio en *GitHub*. La funcionalidad de estos ficheros es:

- README.md: documento que permite mostrar información cuando se abre el repositorio en la página de *GitHub*. Este tipo de documentos se encuentran en la mayoría de los directorios del proyecto para informar que se encuentra en ellos.
- LICENSE: documento que permite mostrar la licencia del proyecto en *GitHub*.
- .gitignore: documento oculto que permite señalar que directorios o ficheros no se quieren subir al repositorio público.

### Documentación

En cuanto a las carpetas se va a empezar comentando la estructura del directorio doc, directorio donde se encuentra toda la documentación del proyecto, que tiene la siguiente composición:

```
/doc/  
└── Diagramas  
└── Latex  
└── Vídeos
```

En el directorio Diagramas se encuentran todos los documentos de diseño realizados para el proyecto, la exportación de estos documentos se encuentra

---

<sup>1</sup>Repositorio: <https://github.com/Josemi/TFM-FIS-IA>

dentro del otro directorio Latex. Para realizar los diseños se han utilizado distintos programas como son DIA [22] y StarUML [7].

En la carpeta Vídeos se encuentran una serie de vídeos que hacen un resumen de la estructura del proyecto, así como muestran alguna ejecución de alguna parte del proyecto.

Por otro lado, la carpeta Latex contiene todos los documentos necesarios para la documentación en L<sup>A</sup>T<sub>E</sub>X de la memoria y de los apéndices del proyecto. Cabe destacar que en la carpeta */doc/Latex/img* se encuentran las exportaciones de los diagramas comentados, además del resto de imágenes usadas en el documento.

## Código

El código de las implementaciones y las pruebas se encuentra en la carpeta */src*, de la cual cuelga los siguientes subdirectorios:

```
/src/
  ├── pruebas
  ├── pos
  ├── python-flujo
  └── visualizarComparacion.ipynb
```

Como se puede observar, hay tres carpetas de código y un fichero *Jupyter Notebook*, las cuales corresponden con:

- **pruebas**: conjunto de pruebas de usadas en *Detectron2* desde la selección del modelo y del *threshold* hasta las pruebas con los problemas con los brazos.
- **pos**: conjunto de ficheros de *Jupyter Notebooks* donde se encuentran las versiones de las posiciones y de la comparación, junto con las pruebas de tiempo de ejecución.
- **python-flujo**: conjunto de ficheros *Python* donde se encuentran todas las versiones implementadas tanto de la posición como de comparación de posiciones. Además, se encuentra el fichero *fishhubuia.py* donde se encuentra la implementación que junto con el fichero *PosicionVF.py* permite la ejecución del cálculo de la posición y comparación en el flujo.
- **visualizarComparacion.ipynb**: fichero *Jupyter Notebook* creado para mostrar el funcionamiento de la comparación y aplicar los cono-

## APÉNDICE C. DOCUMENTACIÓN TÉCNICA DE PROGRAMACIÓN

cimientos de visualización de datos obtenidos durante diversas asignaturas del máster para mostrar información relevante sobre esta. Con esto también se quiere mostrar la capacidad que tiene el sistema implementado.

### C.3. Manual del programador

Como se ha explicado, el código final se encuentra en la carpeta `/src/python-flujo`, donde están todas las versiones de la clase de la posición y todas las versiones de las comparaciones de estas. Además, se encuentra el fichero final que permite, a partir de una clase llamada `Interfaz`, llevar todo el proceso necesario en el flujo: carga del modelo final, liberación de memoria de la GPU, obtención y comparación de las posiciones.

Aun así, como se ha realizado en el desarrollo de este proyecto, se aconseja la implementación de nuevos algoritmos en *Jupyter Notebook* [20], ya que como se ha podido ver en el máster, es una herramienta para desarrollar y ejecutar código muy sencilla y muy usada, sobre todo en la programación en *Python*. Y una vez se haya probado su correcto funcionamiento pasar su implementación a ficheros *Python* (extensión `.py`).

Por último, si se desea continuar con las pruebas realizadas se pueden añadir a la carpeta `/src/pruebas` o a `/src/pos` (para pruebas relacionadas con la duración de la ejecución sobre las versiones ya implementadas) donde se encuentran los distintos *Jupyter Notebooks* con las pruebas implementadas. Se recomienda añadir una carpeta con datos y crear otra para la salida de los algoritmos, como se ha realizado en el proyecto (aunque no se incluyen en el repositorio en *GitHub*).

### C.4. Compilación, instalación y ejecución del proyecto

En este apartado se comenta los requisitos necesarios para ejecutar las implementaciones del proyecto, el proceso de instalación de la herramienta principal sobre la que se sustenta el proyecto, *Detectron2*, y por último la forma de ejecutar las implementaciones.

## C.4. COMPILACIÓN, INSTALACIÓN Y EJECUCIÓN DEL PROYECTO

### Requisitos

Los requisitos, en informática, son las características mínimas necesarias para ejecutar un programa. Estos requisitos pueden ser de dos tipos:

- ***Software*:**

- Consola *Unix* o *macOS*.
- *Python 3*.
- Instalación *drivers* de *CUDA*, versión de 9.2 a 10.2.
- Librerías:
  - *Numpy*.
  - *Pickle*.
  - *Pandas*.
  - *OpenCV (cv2)*.
  - *Garbage Collector (gc)*.
  - Librería de manejo *NVIDIA (pynvml)*.
  - *Matplotlib*.
  - *Seaborn*.
  - *Plotly*.
  - *PyTorch (torch)*.
  - *Torchvision*.
  - *Math*.
  - *Os*.

- ***Hardware*:**

- Tarjeta gráfica con *drivers CUDA* con al menos 1GB de memoria.

Todas las librerías comentadas funcionan con su última versión, excepto *PyTorch*. Con *PyTorch* hay que instalar la versión para los *drivers* de *CUDA* instalados, esto se puede hacer con el comando «`conda install pytorch torchvision cudatoolkit=[versión] -c pytorch`» [26]. Si ya se tiene unos *drivers* instalados se puede comprobar su versión con el comando «`nvcc -version`».

El resto de librerías se pueden instalar normal desde consola con el comando «`pip install [librería]`» o usando *Jupyter Notebook* en una celda de ejecución con el comando «`!pip install [librería]`».

## APÉNDICE C. DOCUMENTACIÓN TÉCNICA DE PROGRAMACIÓN

Los requisitos dados son para ejecutar el código programado, no para lanzar los algoritmos en un flujo.

### Instalación

Una vez se cumplen los requisitos para poder ejecutar las implementaciones, se ha de instalar la herramienta sobre las que se sustenta todo el proyecto, *Detectron2*. Para poder realizar la instalación se puede realizar de dos maneras dependiendo si se prefiere clonar el repositorio o instalar solo lo necesario [31].

Para realizar la instalación a partir del clonado se ha de ejecutar primero el comando de clonado del repositorio en *GithHub* con «`git clone https://github.com/facebookresearch/detectron2.git`». Una vez ha terminado la clonación del repositorio se ha de instalar, estando en el directorio donde se ha clonado, la herramienta con el comando «`python -m pip install -e detectron2`».

Por otro lado, si se desea realizar la instalación sin clonar el repositorio entero se puede hacer a partir de los enlaces del documento de instalación disponible en el repositorio [31]. En este documento hay una tabla donde muestran los distintos comandos de instalación dependiendo de la versión *CUDA* y *PyTorch* instalada. Por ejemplo, para la versión 10.2 de *CUDA* y la versión de *PyTorch* 1.5 se utilizaría el comando: «`python -m pip install detectron2 -f https://dl.fbaipublicfiles.com/detectron2/wheels/cu102/torch1.5/index.html`».

Cabe destacar, que existe un *notebook*<sup>2</sup> en *Google Colab* donde se puede probar la herramienta sin realizar ninguna instalación en el dispositivo local.

### Ejecución

Como ya se ha explicado, los códigos implementados están almacenados en dos tipos distintos de ficheros: *Python* y *Jupyter Notebook*. Si se desea ejecutar lo código en *Python* se han de importar para su posterior uso. En cambio, los ficheros de *Jupyter Notebook* se pueden ejecutar desde ese mismo entorno simplemente ejecutando las celdas en el orden de aparición.

Si se quiere usar el código de la última versión para el flujo simplemente se ha de descargar la última *release* y en el código donde se realizan las operaciones del flujo importar la clase *Interfaz* dentro del fichero *fishu-*

---

<sup>2</sup>*Google Colab Detectron2*

*buia.py* y llamar a sus funciones para realizar las estimaciones de la posición, comparar posiciones y liberar la memoria.

Cabe decir que es necesario tener espacio libre en la memoria de la GPU para ejecutar cualquier operación con el modelo, ya sea cargarlo o estimar con él. Es importante este espacio porque sino *CUDA* devuelve un error por falta de espacio, es por ello que se implementó una función que libera espacio de la memoria para poder ejecutar más de un modelo.

## C.5. Pruebas del sistema

Como se ha podido observar, el proyecto ha tenido dos fases bien diferenciadas, una primera fase donde se ha realizado primero un estudio del estado del arte de las herramientas de visión artificial capaces de estimar la posición de una persona y después, una vez se había elegido la herramienta, un segundo estudio del estado del arte de los distintos modelos proporcionados por la herramienta, con el posterior estudio de sus parámetros y salidas. Y una segunda fase, donde a partir del estudio realizado en la fase anterior, se realizaron distintas implementaciones para la extracción de características de la salida del modelo seleccionado (clase de la posición) y la comparación de posiciones.

El desarrollo de estas dos fases se ha seguido bajo una continua evaluación por pruebas con distinta finalidad, en la primera fase se usaron pruebas que permitían conocer y evaluar los distintos modelos de *Detectron2* y evaluar el efecto del *threshold* sobre las estimaciones. Mientras que en la segunda fase las pruebas se centraron en comprobar el correcto funcionamiento de la extracción de características y de la comparación de las posiciones obtenidas. Dentro de estas pruebas para comprobar el funcionamiento de las implementaciones se dividieron en dos ramas dependiendo de con qué datos se trabajaba, ya que se han implementado pruebas de funcionamiento sobre vídeos parecidos a los que se pueden realizar a una rehabilitación de pacientes con *Parkinson*, pero además se han utilizado imágenes especiales que pueden llevar a error en la comparación de posiciones (como se han podido ver a lo largo del documento, especialmente en las pruebas con los brazos). Además, sobre esta segunda fase se han realizado otro tipo de pruebas, estas son las relacionadas con la duración de la ejecución de los algoritmos y el cálculo del número de *workers* en la parallelización del flujo.

Como se puede observar, este tipo de pruebas (que como se ha comentado se encuentran en las carpetas */src/pos* y */src/pruebas*) no son pruebas que

## **APÉNDICE C. DOCUMENTACIÓN TÉCNICA DE PROGRAMACIÓN**

puedan dar una evaluación objetiva sobre el funcionamiento del sistema como puede ser la evaluación de un algoritmo de aprendizaje supervisado o no supervisado. Aun así, estas pruebas han sido esencial en todas las fases del desarrollo para poder seleccionar el modelo correcto, seleccionar los parámetros que mejor se ajusten al problema planteado y poder comprobar el funcionamiento de las implementaciones realizadas.

*Apéndice D*

---

## **Documentación de usuario**

---

- D.1. Introducción**
- D.2. Requisitos de usuarios**
- D.3. Instalación**
- D.4. Manual del usuario**



---

## Bibliografía

---

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jo-nathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- [2] Formación audiovisual. El cine y los fotogramas por segundo. <https://www.formacionaudiovisual.com/blog/cine-y-tv/el-cine-y-los-fotogramas-por-segundo-fps/>, jan 2017.
- [3] Michaël Bonenfant, Denis Laurendeau, Alexis Fortin-Côté, Philippe Cardou, and Clément Gosselin. A computer vision system for virtual rehabilitation. <https://doi.org/10.1109/CRV.2017.30>, may 2017.
- [4] Yao-Jen Chang, Shu-Fang Chen, and Jun-Da Huang. A kinect-based system for physical rehabilitation: A pilot study for young adults with motor disabilities. <https://doi.org/10.1016/j.ridd.2011.07.002>, dec 2011.
- [5] Mayo Clinic. Enfermedad de parkinson. <https://www.mayoclinic.org/es-es/diseases-conditions/parkinsons-disease/symptoms-causes/syc-20376055#:~:text=La%20enfermedad%20de%>

- 20Parkinson%20es , rigidez%20o%20disminuci%C3%B3n%20del%20movimiento .
- [6] Mayo Clinic. Trastornos del movimiento. <https://www.mayoclinic.org/es-es/diseases-conditions/movement-disorders/symptoms-causes/syc-20363893#:~:text=El%20t%C3%A9rmino%20de%20trastornos%20del%20movimiento,pueden%20ser%20voluntarios%20o%20involuntarios.>
  - [7] MKLabs Co. Staruml 3. <http://staruml.io/>, 2020.
  - [8] Organización Mundial de la Salud. Enfermedades crónicas. [https://www.who.int/topics/chronic\\_diseases/es/](https://www.who.int/topics/chronic_diseases/es/).
  - [9] DeepLizard. Zero padding in convolutional neural networks explained. [https://deeplizard.com/learn/video/qSTv\\_m-KFk0](https://deeplizard.com/learn/video/qSTv_m-KFk0).
  - [10] Curemos el Parkinson. Día internacional del parkinson 2020: mitos e investigación. <https://conoceelparkinson.org/cuidados/dia-internacional-parkinson-2020/>, apr 2020.
  - [11] Facebook Engineering. Introducing big basin: Our next-generation ai hardware. <https://engineering.fb.com/data-center-engineering/introducing-big-basin-our-next-generation-ai-hardware/>, mar 2017.
  - [12] Baki Er. Microsoft presents : Deep residual networks. <https://medium.com/@bakiiii/microsoft-presents-deep-residual-networks-d0ebd3fe5887>, aug 2016.
  - [13] Parkinson Federación Española. Conoce la enfermedad. <https://www.esparkinson.es/espacio-parkinson/conocer-la-enfermedad/>.
  - [14] Yoav Freund and Robert E. Schapire. A short introduction to boosting, sep 1999.
  - [15] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. <https://github.com/facebookresearch/detectron>, 2018.
  - [16] D. González-Ortega, F.J. Díaz-Pernas, M. Martínez-Zarzuela, and M. Antón-Rodríguez. A kinect-based system for cognitive rehabilitation exercises monitoring. <https://doi.org/10.1016/j.cmpb.2013.10.014>, feb 2014.

- [17] Ming-Kuei Hu. Visual pattern recognition by moment invariants. <https://doi.org/10.1109/ICORR.2011.5975421>, jul 2011.
- [18] ides. Gradior. <https://ides.es/gradior>, 2014.
- [19] Salman Khan, Hossein Rahmani, Syed Afaq Ali Shah, and Mohammed Bennamoun. *A Guide to Convolutional Neural Networks for Computer Vision*. Gérard Medioni & Sven Dickinson. Morgan & Claypool, 2018.
- [20] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.
- [21] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. [http://openaccess.thecvf.com/content\\_cvpr\\_2017/html/Lin\\_Feature\\_Pyramid\\_Networks\\_CVPR\\_2017\\_paper.html](http://openaccess.thecvf.com/content_cvpr_2017/html/Lin_Feature_Pyramid_Networks_CVPR_2017_paper.html), 2017.
- [22] Steffen Macke. Dia diagram editor. <http://dia-installer.de/>.
- [23] MedlinePlus. Enfermedad de parkinson. <https://medlineplus.gov/spanish/parkinsonsdisease.html>.
- [24] Nvidia. Nvlink y nvswitch: Los elementos fundamentales de la comunicación multi-gpu avanzada. <https://www.nvidia.com/es-es/data-center/nvlink/>.
- [25] OnlineMSchool. Ángulo entre vectores. <http://es.onlinemschool.com/math/library/vector/angl/#:~:text=%C3%81ngulo%20entre%20dos%20vectores%20trazados,direcci%C3%B3n%20con%20el%20otro%20vector.>
- [26] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.
- [27] JPND research. ¿qué es una enfermedad neurodegenerativa? <https://www.neurodegenerationresearch.eu/es/que-es-una-enfermedad-neurodegenerativa/>.

- [28] Wikipedia. Visión artificial — wikipedia, la enciclopedia libre. [https://es.wikipedia.org/w/index.php?title=Visi%C3%B3n\\_artificial&oldid=120821222](https://es.wikipedia.org/w/index.php?title=Visi%C3%B3n_artificial&oldid=120821222), 2019.
- [29] Wikipedia. Kinect — wikipedia, la enciclopedia libre. <https://es.wikipedia.org/w/index.php?title=Kinect&oldid=125711991>, 2020.
- [30] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. <https://github.com/facebookresearch/detectron2/blob/master/INSTALL.md>, 2019.
- [31] Yuxin Wu, Julius Simonelli, Jeremy Fix, and Anshul Rai. Instalación detectron2. <https://github.com/facebookresearch/detectron2/blob/master/INSTALL.md>, 2020.
- [32] Sepp Hochreiter y Jürgen Schmidhuber. Long short-term memory, nov 1997.
- [33] José Zariffa and John D. Steeves. Computer vision-based classification of hand grip variations in neurorehabilitation. <https://doi.org/10.1109/ICORR.2011.5975421>, jul 2011.