



Desarrollo de un algoritmo de machine learning para el control de una prótesis de mano mediante señales electromiográficas

PROYECTO

presentado para optar
al Título de Grado en Ingeniería Biomédica por
Carlos Martínez de la Fuente
bajo la supervisión de
José Sebastián Gutiérrez Calderón

Donostia-San Sebastián, junio de 2025



Tecnun
Universidad
de Navarra

ESCUELA DE INGENIERÍA
INGENIARITZA ESKOLA
SCHOOL OF ENGINEERING



Tecnun
Universidad
de Navarra

ESCUELA DE INGENIERÍA
INGENIARITZA ESKOLA
SCHOOL OF ENGINEERING

Proyecto Fin de Grado

INGENIERÍA BIOMÉDICA

Desarrollo de un algoritmo de machine learning para el control de una prótesis de mano mediante señales electromiográficas

Carlos Martínez de la Fuente

Donostia-San Sebastián, junio de 2025

Resumen

Este Proyecto de Fin de Grado desarrolló un sistema de control de una prótesis de mano basado en algoritmos de *Machine Learning* (ML). Para el modelo del entrenamiento se confeccionó una base de datos a partir de 52 minutos de grabación que unía señales obtenidas mediante electromiografía (EMG) superficial de 5 canales a nivel del antebrazo, obtenidas con un sistema BrainAmp; y un modelo cinemático de la mano obtenido a partir de vídeos y visión artificial con MediaPipe. Fue necesario hacer un filtrado de ambas señales, en el caso de la EMG, se emplearon filtros Notch a 50 Hz y pasa-banda desde 10 a 450 Hz junto con un remuestreo a 1024 Hz; en el caso del modelo cinemático, se capturó en vídeo a 32 fotogramas por segundo (FPS) para una relación de número entero con las señales EMG y además se aplicó un filtro Savitzky-Golay de segundo grado con una ventana de 33 datos para suavizar las coordenadas extraídas por el modelo de visión artificial. Se implementaron modelos de redes neuronales artificiales (ANN), convolucionales (CNN) y *transformers* con atención entrenados mediante aprendizaje supervisado optimizando el error absoluto medio (MAE) y empleando la raíz del error cuadrático medio (RMSE) como indicador de evaluación. Se concluyó que el modelo *transformer* con atención (que obtuvo un MAE de 58° en test) fue el único capaz de abstraer la naturaleza del movimiento de la mano. Esto se comprobó al representar gráficamente los resultados de las predicciones obtenidas en los test, que mostraron como los modelos sin atención tendieron a situar puntos inconexos alrededor de un valor promedio que redujo artificialmente el error (ANN: 9°; CNN: 2°). Por tanto se concluyó que la atención fue un factor fundamental en la correcta lectura de las señales EMG. Finalmente, se llevó a cabo una implementación de comunicación con un microcontrolador tipo Arduino desde Python y a través de puerto serie, que comunicaría el modelo obtenido con una prótesis.

Abstract

This Final Degree Project developed a control system for a hand prosthesis based on *Machine Learning* (ML) algorithms. For the training model, a database was created from 52 minutes of recording that combined signals obtained via 5-channel surface electromyography (EMG) at the forearm, recorded with a BrainAmp system; and a kinematic model of the hand obtained from videos using computer vision with MediaPipe. Both signals required filtering: for the EMG, 50 Hz notch and 10–450 Hz band-pass filters were applied along with resampling to 1024 Hz; for the kinematic model, videos were captured at 32 frames per second (FPS) to achieve an integer ratio with the EMG signals, and a second-degree Savitzky-Golay filter with a window of 33 data points was applied to smooth the coordinates extracted by the computer vision model. Artificial neural networks (ANNs), convolutional neural networks (CNNs), and attention-based *transformers* were implemented and trained via supervised learning, optimizing the mean absolute error (MAE) and using the root mean square error (RMSE) as an evaluation metric. It was concluded that the attention-based *transformer* model (which achieved a test MAE of 58°) was the only one capable of capturing the nature of hand movement. This was verified by graphically representing the prediction results on the test set, which showed that models without attention tended to place disconnected points around an average value, artificially reducing the error (ANN: 9°; CNN: 2°). Therefore, attention was concluded to be a fundamental factor in correctly reading EMG signals. Finally, communication with an Arduino-type microcontroller from Python via serial port was implemented, allowing the trained model to interface with a prosthesis.

Agradecimientos

Desde pequeño siempre he sentido una especial devoción por la informática y una pasión por la robótica y la programación. Con el tiempo, también me enamoré de la biología y de la máquina perfecta, el cuerpo humano. Es por eso que me siento profundamente agradecido a mi director del proyecto, José Sebastián, que me apoyó para desarrollarlo.

Quiero agradecer también a todos los profesores que me han acompañado durante esta etapa universitaria y en especial al departamento de ingeniería de tejidos, Ana, Jacobo y Javi; a Mireya y Ane; y a Javier Díaz, sin el cual no habría sido posible la toma de EMG.

A mi familia que me ha visto crecer durante estos años y a mí madre en especial, que ha terminado aprendiendo como se entrena una red neuronal por *machine learning*.

A mis amigos del “Objetivo”, por que podamos hacer muchos más viajes. A Ignacio por “alegrarme” los días en clase. Eskerrik asko Iñigori bere argazkiagatik (Figura 4.7). Eta Peiori, que vivió en un tren.

Y a la “Bioficina”. Ojalá poder volver y tomar un café más. A María, que me enseñó a atar un nudo de ocho. A Ana, por el café con leche con hielo con azúcar. A Terese por los trozos de silicona. A Gontxi por sus moscas. A Peter por escuchar. Y al resto, muchas gracias por todo.

*No me asusta la
inteligencia artificial.*

*Me asusta la
estupidez natural.*

Isaac Asimov

Índice general

1. Introducción	1
1.1. Objetivos	2
1.2. Estructura de la memoria	3
2. Estado del arte	4
3. Metodología y herramientas	13
3.1. Procesado de la base de datos	15
3.2. Desarrollo metodológico	16
4. Desarrollo	18
4.1. Estudio previo y calibración señales	18
4.1.1. Reconocimiento de imagen	18
4.1.2. Señales de EMG	24
4.1.3. Setup para grabación de imagen y EMG simultánea	34
4.2. Confección del set de datos	36
4.2.1. Procesado de vídeo	37
4.2.2. Procesado de señal EMG	39
4.2.3. Fusión de datos	41
4.3. Obtención del modelo	42
4.3.1. Selección de variables	43
4.3.2. Entrenamiento del modelo	43
4.3.3. Evaluación de los modelos	45
4.3.4. Estudio de la evolución del modelo	46
4.4. Conectividad con Arduino	46

5. Resultados y discusión	50
5.1. Evaluación de la base de datos	50
5.2. Evaluación de los modelos	51
5.3. Evaluación de escalabilidad del estudio	58
6. Conclusiones	60
6.1. Conclusiones fundamentales	60
6.2. Limitaciones técnicas y futuras líneas de estudio	61
Apéndices	65
A. Entorno de Conda y dependencias de Python	66
B. Tutorial para usar BrainVision Recorder	67
B.1. Creación del entorno de trabajo	68
B.2. Creación del <i>Display Montage</i>	69
B.3. Comprobar impedancias y grabar	69
C. Tutorial para usar BrainVision Analyzer	76
C.1. Creación del entorno de trabajo	77
C.2. Procesado de la señal	78
D. Código y funciones de Python	88
D.1. Funciones y script para grabar	88
D.2. Funciones relacionadas con visión artificial	89
D.3. Funciones relacionadas con el tratamiento del vídeo	91
D.4. Funciones relacionadas con el tratamiento de EMG	93
D.5. Funciones relacionadas con la fusión del <i>dataset</i>	94

Índice de figuras

2.1. Representación habitual de una red neuronal con n neuronas de entrada y k de salida.	7
2.2. Funciones de activación: ReLU, Leaky ReLU y Logistic.	7
2.3. Expresión matemática que regula la activación de una neurona dentro de una red.	8
3.1. Esqueleto de la mano obtenido mediante la librería mediapipe con el número de articulación.	15
3.2. Infografía de la relación entre Conda, Miniconda y Anaconda como se ilustra en [22].	16
3.3. Esquema del desarrollo metodológico seguido en este Proyecto de Fin de Grado (PFG)	17
4.1. Capturas de la confianza para encontrar la mano (<i>tracking confidence</i>) del modelo cuando la palma de la mano se encuentra mirando hacia el objetivo.	19
4.2. Capturas de la confianza para encontrar la mano (<i>tracking confidence</i>) del modelo cuando el dorso de la mano se encuentra mirando hacia el objetivo.	20
4.3. Posición en la coordenada x de los dedos a lo largo del tiempo al abrir y cerrar la mano.	21
4.4. Comparación de la coordenada z de la punta del meñique antes y después de usar el filtro de Savitzky-Golay	23
4.6. Posición de los electrodos en la ubicación muscular en el brazo izquierdo de un voluntario.	25
4.7. Posición de los electrodos en “pulsera” en el brazo derecho de un voluntario.	25
4.8. Imagen del <i>setup</i> experimental con los electrodos posicionados y el soporte para el brazo situado. A la izquierda se ve el amplificador de instrumentación conectado.	26
4.9. Imagen del sistema de conexiones para mandar señales manuales durante el experimento.	26
4.10. Captura de la pestaña de impedancias de BrainVision Recorder.	27
4.11. Comparativa entre señal cruda y después del filtro de frecuencias que se ve en la Tabla 4.2.	28
4.12. Comparativa de la Transformada de Fourier (FT) entre la apertura y cierre de la mano para los pares de electrodos según la disposición muscular de la Figura 4.6.	30

4.13. Comparativa de la FT entre la apertura y cierre de la mano para los pares de electrodos siguiendo la disposición de “pulsera” de la Figura 4.7.	31
4.14. Estudio del ruido introducido en cada canal a causa de una muñeca en tensión.	33
4.15. <i>Wireframe</i> de la mano con los <i>landmarks</i> . Se indica también la nomenclatura empleada para nombrar los ángulos extraídos.	39
4.16. Representación del <i>pipeline</i> seguido habitualmente en procesos de aprendizaje automático. .	42
5.1. Comparación del <i>Mean Absolute Error</i> (MAE) en proporción con el rango de flexión de la articulación entre los dos modelos para cada tipo de <i>dataset</i>	53
5.2. MAE obtenido en proporción con el rango de flexión de las articulaciones de la mano . .	54
5.3. Estudio de la presencia de deriva en las predicciones y contraste de la precisión para la predicción entre Red Neuronal Convolutacional (CNN) cuando se da información sobre la posición anterior y Red Neuronal Artificial (ANN) cuando solo recibe la señal de Electromiografía (EMG).	56
5.4. Estudio de la precisión de la predicción del modelo transformador con atención cuando se le pasa la señal de EMG comparado con el movimiento real del dedo para datos nunca vistos.	57
5.5. Evolución de los errores a medida que se entrena el modelo <i>transformer</i> con atención con más vídeos. Los vídeos (identificado como número de modelo) siguen el orden declarado en la Tabla 5.1, y con franjas de color se expresan los movimientos-tipo presentes en dichos vídeos.	58

Índice de tablas

2.1. Comparación entre los distintos modelos de red neuronal empleados en el estudio.	6
2.2. Comparación entre las distintas tecnologías de captura de movimiento disponibles.	10
4.1. Músculos elegidos para el estudio y sus electrodos junto al dedo que afecta.	24
4.2. Parámetros para el filtrado de la señal cruda [13].	28
4.3. Comparativa de los <i>p.values</i> obtenidos en la Figura 4.12 y Figura 4.13 ordenados de mayor a menor relevancia estadística.	29
4.4. Ejemplo de fichero .csv donde se almacenan los datos de las coordenadas de la mano.	39
4.5. Ejemplo de tabla en la que se guardan los valores de los impulsos eléctricos de la EMG. .	41
4.6. Estructura final dentro del .csv de la base de datos.	42
4.7. Posibles combinaciones de los datos para entrenar al modelo.	43
5.1. Características de los vídeos adquiridos durante la confección de la base de datos.	51
5.2. Valores obtenidos de MAE y <i>Root Mean Squared Error</i> (RMSE) al final del entrenamiento del modelo (validación).	52
5.3. Valores obtenidos de MAE y RMSE al probar (test) los modelos entrenados.	52

Lista de acrónimos

ANN Red Neuronal Artificial.

API Interfaz de Programación de Aplicaciones.

CNN Red Neuronal Convolucionar.

DIP Articulación Interfalángica Distal.

EMG Electromiografía.

FPS Fotogramas por Segundo.

FT Transformada de Fourier.

IA Inteligencia Artificial.

MAE *Mean Absolute Error.*

MCP Articulación Metacarpofalángica.

ML *Machine Learning.*

MSE *Mean Squared Error.*

PFG Proyecto de Fin de Grado.

PIP Articulación Interfalángica Proximal.

RMSE *Root Mean Squared Error.*

1 Introducción

La falta de un miembro ya sea superior o inferior genera una gran limitación en la vida cotidiana de los pacientes que las padecen, por lo que la industria trabaja para que las ortoprótesis puedan facilitar todo lo posible la autonomía de los mismos.

Muchas pueden ser las causas de estas lesiones, y por tanto el tipo de ortoprótesis. Puede haberse debido a una amputación relacionada por ejemplo con accidentes de tráfico, enfermedades cardiovasculares o diabetes, o bien por malformaciones congénitas que pueden afectar no solo a la estética de la extremidad, sino también a la funcionalidad de la misma.

La prótesis estética, no funcional, suelen ser menos costosas, pero si en realidad se quiere conseguir la autonomía del paciente, se ha de apostar por las prótesis funcionales, que sustituyan las capacidades perdidas, y por supuesto, hacerlo de tal forma que tampoco genere un gasto excesivo, al paciente.

Para paliar la discapacidad generada en los miembros superiores, se pueden inferir actuaciones a nivel transradial, transhumeral o en las articulaciones. Esto hace necesaria una individualización de la prótesis que sustituya o restablezca las capacidades funcionales lesionadas que el mercado ortoprotésico soluciona con las ortoprótesis fabricadas en serie, con una mínima adaptación, o las prótesis fabricadas a medida.

En la fabricación de una prótesis, intervienen tanto la estética como la ergonomía junto con los materiales utilizados, de manera que se consiga con todo ello un funcionamiento y apariencia lo más similar posible al miembro real, lo que sin duda conlleva un bienestar no sólo físico sino también psicológico del paciente.

Diversos factores influyen en el correcto funcionamiento de una prótesis, como son el procesamiento de la señal, el reconocimiento de patrones, la activación de los actuadores, el propio diseño de la prótesis. Estos factores ralentizan y encarecen el proceso de fabricación.

En la actualidad, la industria fabricante de prótesis se decanta por dos opciones:

Por un lado, se encuentran las prótesis fabricadas utilizando la impresión 3D, que consiguen una

aproximación con un prototipado más económico y una adaptación a la fisionomía del paciente máxima.

Por otro, las prótesis funcionales que a través de variables de los distintos formatos de control, como pueden ser la electroencefalografía, la electrooculografía, los comandos de voz o la EMG superficial, logren una prótesis activa con movimientos similares a la extremidad biológica. En este trabajo se ha utilizado esta última variable para mejorar el movimiento de la prótesis, con un sistema que pretende abaratar el coste de producción, desarrollando sistemas de control precisos.

El análisis de las variables anteriormente citadas, se puede hacer desde el dominio del tiempo o de la frecuencia, entre otros. Generalmente se usa el dominio del tiempo por la facilidad para trabajar con él y la poca carga computacional que supone, sobre todo al tener en cuenta que esto deberá ser llevado a cabo por un chip integrado en la prótesis cuya capacidad en recursos computacionales es limitada. Sin embargo, en otros ámbitos se da más importancia al dominio de las frecuencias, ya que este ofrece la posibilidad de analizar los rangos de frecuencias excitados ofreciendo una información que el otro no consigue. [1].

En la actualidad se ha logrado que la relación entre la señal y los actuadores, realizado a través de un algoritmo de control, se abarate gracias al *Machine Learning* (ML).

Este PFG busca como tantos otros abaratar costes en el caso concreto de prótesis de mano y aumentar su accesibilidad. Esto se tratará de lograr empleando algoritmos de ML junto con la confección de un set de datos partiendo de una señal electromiográfica y un modelo cinemático de la mano.

1.1 Objetivos

Como objetivo principal del PFG se busca desarrollar un algoritmo de ML que dada una señal de EMG sea capaz de predecir el movimiento de los dedos de la mano derecha. De este objetivo principal derivan otros objetivos secundarios con la intención de recoger y documentar los procedimientos empleados durante el PFG y que justifiquen las decisiones de diseño tomadas durante el mismo. Se pueden definir estos objetivos como los siguientes:

- Obtener protocolos para obtener señales de EMG y un modelo cinemático en paralelo.
- Lograr una base de datos con la que entrenar un modelo de ML y que recoja los datos de EMG junto con las coordenadas del modelo cinemático.
- Preparar *scripts* y aplicaciones que automatizan los procedimientos de captura y combinación de señales.

- Obtener un modelo regresivo que asocie los datos del modelo cinemático con los impulsos eléctricos de las señales obtenidas de EMG.
- Estudiar la calidad del modelo generado.
- Encontrar un tamaño muestral que genere un modelo viable para entrenamiento por ML.
- Preparar una conexión que permita emplear el modelo obtenido en una placa controladora para controlar un sistema real.

1.2 Estructura de la memoria

A continuación, se describe de forma resumida los contenidos de los apartados del PFG:

- **Capítulo 2 Estado del arte.** En este capítulo se describe la situación actual del control de prótesis mediante señal de EMG y modelos de Inteligencia Artificial (IA) generados por ML. Además se describe de manera superficial el funcionamiento de una red neuronal típica, todo ello para contextualizar el presente trabajo.
- **Capítulo 3 Metodología y herramientas.** En este capítulo se describen y explican las herramientas empleadas durante el PFG tanto de *hardware* como de *software*, a fin de entender el funcionamiento y la decisión de uso. También se establecen las instrucciones que se deberán desarrollar durante el proyecto.
- **Capítulo 4 Desarrollo.** En este capítulo se recoge todo el proceso del PFG y las tareas realizadas hasta lograr el modelo de IA objetivo.
- **Capítulo 5 Resultados y discusión.** En este capítulo se recogen los resultados de validación del modelo objetivo.
- **Capítulo 6 Conclusiones.** En este capítulo se analizan los resultados del capítulo anterior y se reflexiona sobre el alcance del estudio.

Esta memoria posee vínculos programados en ella que al ser visualizada en lectores de PDF compatibles, dejará pulsar elementos como las menciones a figuras, tablas, secciones o citas (p. ej Figura 2.1 o Apéndice C) para saltar directamente a la referencia. También es posible pulsar en el índice para desplazarse a la sección. Por último, en el cuerpo del documento también es posible pulsar en los números de página de los encabezados para regresar al índice.

2 Estado del arte

El desarrollo de sistemas de aprendizaje automático en la generación de modelos predictivos, ha supuesto un gran avance sobre todo en el procesamiento de señales, que ha permitido eliminar los “ruido” generados en la propia técnica, y conseguir así una gran eficacia en generación del modelo predictivo.

En el ámbito de la medicina, la EMG, técnica por la cual se obtienen señales de actividad eléctrica producida por los músculos esqueléticos durante la contracción o el reposo, generada por los potenciales de acción de las fibras musculares al activarse, ofrece valiosa información sobre la función neuromuscular del paciente.

La adquisición de la señal de EMG, puede llevarse a cabo, a nivel intramuscular, mediante una aguja que es insertada en el músculo directamente, o bien a través de electrodos en la superficie de la piel. La primera ha de hacerse por personal especializado, y supone para el paciente un método invasivo y molesto, sin embargo, los registros de señal presentan poco ruido. La obtención de señal a partir de la superficie de la epidermis utilizando parches autoadhesivos, presenta la ventaja de no necesitar profesionales especializados, y ser más cómoda para el paciente, en su contra, puede captar señales de los músculos vecinos (interferencia o “cross-talk”), obteniendo señales con más ruido.

Debido a este ruido, la obtención de señales a través de EMG superficial tienen una eficiencia media del 70 %, de forma que para neutralizarlo se utilizan diversos métodos de procesamiento de los datos de las señales, que a través de algoritmos o programas con la capacidad de aprender de patrones, pueden obtenerse registros con una eficacia de reconocimiento del movimiento de una 99 %. Por ello, las tecnologías de aprendizaje automático junto con la EMG se aplican en diferentes campos, como el de los brazos robóticos, reconocimiento de voz o análisis biomecánico de la marcha entre otras aplicaciones. [2].

El aprendizaje automático también conocido como ML, crea patrones a partir de datos, generando a su vez un modelo predictivo, que devuelve la respuesta esperada a partir de la serie de datos introducida.

Según la naturaleza de la variable que se busca predecir, el modelo predictivo puede ser de clasificación, cuando la variable es cualitativa, o de regresión, en el caso de variables cuantitativas. En el primer caso, el modelo predice al grupo al que pertenecerá una variable nueva nunca vista en los datos de entrenamiento, mientras que, en el segundo caso, el modelo arrojará un valor numérico.

El proceso de ML consiste en iterar ajustando parámetros para reducir al máximo el error en la respuesta. Se distinguen tres vías de aprendizaje del modelo: La primera es el aprendizaje supervisado en el entrenamiento de la predicción en estos modelos, mediante la asignación de etiquetas que relaciona la respuesta correcta propuesta por el modelo, con la variable. Este modelo puede utilizarse tanto para los modelos de clasificación como para los de regresión.

El aprendizaje no supervisado, es aquel en el que el propio algoritmo trabaja con datos no etiquetados y busca la mejor manera de agruparlos en clústeres. Esta aproximación funciona mejor en los modelos de clasificación o de reducción de dimensionalidad.

Por último, la tercera vía de entrenamiento del algoritmo es a través del aprendizaje por refuerzo. El modelo aprende en “tiempo real” por prueba y error recibiendo recompensas o penalizaciones en función de sus acciones, por lo que siempre buscará repetir aquello que ha dado resultado positivo. Estos modelos son ideales para aprender y evolucionar junto a un sujeto. Sin embargo, pueden desestabilizarse si no se cuida su aprendizaje.

Una estrategia muy extendida y útil es entrenar un modelo supervisado y luego continuar con su aprendizaje por refuerzo. Esto abre las puertas a prótesis personalizadas con capacidad de adaptarse al paciente con el tiempo.

Una subcategoría de ML son las redes neuronales, que se caracteriza por asemejarse al mecanismo de funcionamiento de las neuronas en el cerebro orgánico y con una capacidad excelente para encontrar relaciones no lineales entre los datos de entrada y salida, con aplicación en áreas donde se utilizan grandes cantidades de datos y el objetivo principal es inferir patrones a partir de ellos. Las ventajas de utilizar redes neuronales en lugar de trabajo manual, se centra en las áreas de precisión, velocidad de procesamiento, tolerancia a fallos y rendimiento entre otras, donde la red neuronal es claramente superior al modelo manual.

Existen a su vez redes neuronales de muchos tipos distintos: convolucionales, recurrentes, *feed-forward*, perceptrones o *transformers*...[3]. Este tipo de modelos son idóneos para emular las relaciones que se dan en un cerebro orgánico, puesto que se asimila al procesamiento de las señales biológicas. Entre los más habituales y en los que se centró el estudio estuvieron las CNN, ANN y los *transformers* con atención. Estos modelos presentan diferentes características y usos que se recogen en la Tabla 2.1 a continuación.

Tabla 2.1: Comparación entre los distintos modelos de red neuronal empleados en el estudio.

Característica	ANN	CNN	Transformer (con atención)
Arquitectura	Capas totalmente conectadas	Capas convolucionales (filtros locales)	Capas de auto-atención con codificación posicional
Supuesto de entrada	No supone estructura espacial ni secuencial	Supone correlación espacial local	Supone entrada secuencial; modela relaciones a larga distancia
Tipo de entrada	Datos tabulares o aplanados	Datos en forma de rejilla	Datos secuenciales
Operación principal	Multiplicaciones matriciales	Convolución + pooling	Auto-atención + red feedforward
Campo receptivo	Global, pero no específico	Local (aumenta con la profundidad)	Global desde el principio
Conciencia posicional	Ninguna	Implícita por la estructura de la red	Explícita mediante codificación posicional
Enfoque	Aprende patrones arbitrarios; entrada como vector plano	Aprende patrones locales con creciente abstracción	Aprende relaciones contextuales entre toda la entrada
Escalabilidad	Limitada (sobreajuste fácil)	Escala bien grandes volúmenes de datos	Requiere muchos recursos de cómputo y memoria
Parámetros compartidos	Ninguno, pesos únicos	Sí, mismo filtro para diferentes posiciones	Sí, se comparte el mecanismo de atención

En cuanto a la estructura de la red neuronal típica que se puede ver en la Figura 2.1. Esta estructura particular se corresponde con la de una CNN. Esta representación se puede extender al resto de modelos mencionados en la Tabla 2.1 como bloques básicos de construcción. Es al aplicar sistemas de atención que se emplea dicha estructura (Figura 2.1) como bloques para formar una estructura más compleja.

En este esquema se representan n neuronas de entrada que recibirán (por lo general) $n - 1$ variables de entrada (como por ej. registros de EMG). Estas neuronas de entrada se activarán según el valor y pasarán a la siguiente capa sus valores. Las siguientes tres capas azules se conocen como ocultas (*hidden layers*) y en este caso tienen m neuronas siendo m mayor que n . Aunque en la representación aparecen tres capas, los modelos reales pueden incluir muchas más. Son estas capas (convolucionales en este caso) en las que se logra la no linealidad de los datos y permite encontrar patrones implícitos que de ninguna otra forma se podrían encontrar.

Finalmente se llega a la capa de salida con k neuronas. Este valor k debe coincidir con las variables de salida del problema, por ejemplo, k articulaciones interfalángicas sobre las que se miden los ángulos de flexión.

En cuanto a la matemática que explica el funcionamiento de estas neuronas, nos encontramos con una neurona de entrada (p. ej. x_1) que se excitará según la entrada que reciba y su función de activación (σ). Existen diversas funciones σ como *ReLU*, *Leaky ReLU* o *Logistic*, que regulan el umbral de activación de la neurona y la cantidad de señal pasada a la siguiente capa. Esta función se puede configurar según el caso aplicado y el objetivo buscado. En la Figura 2.2 se ven las tres anteriores.

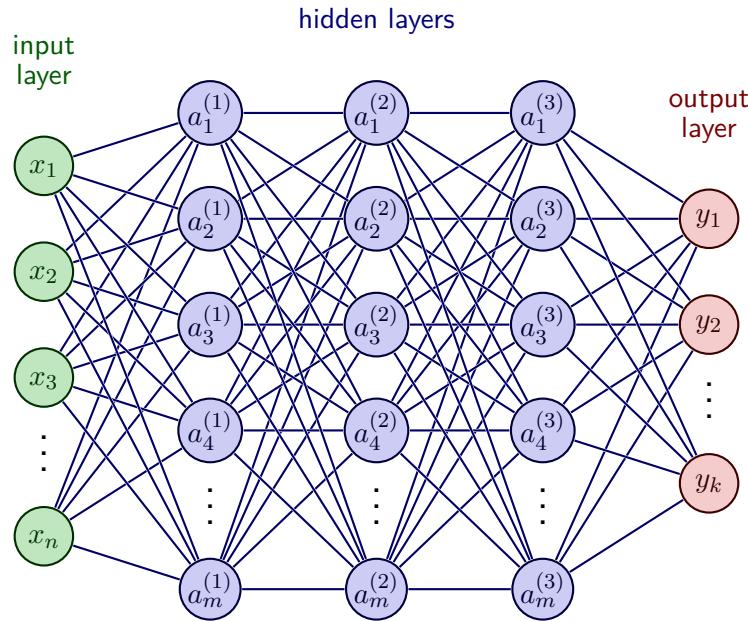


Figura 2.1: Representación habitual de una red neuronal con n neuronas de entrada y k de salida.

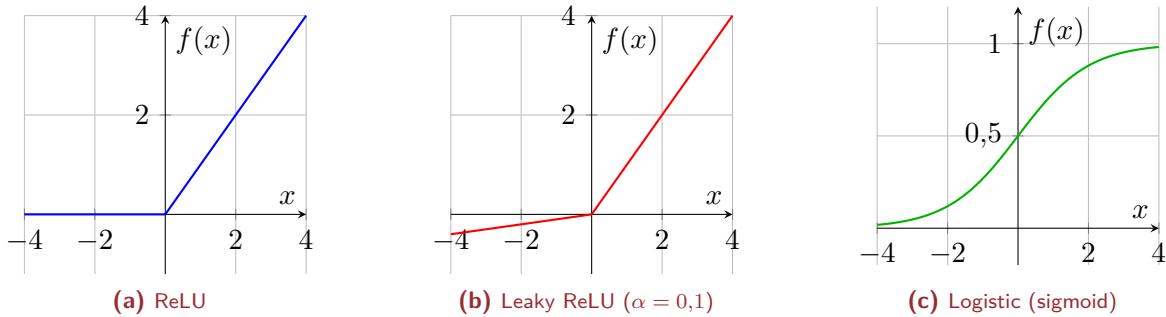


Figura 2.2: Funciones de activación: ReLU, Leaky ReLU y Logistic.

Después de esto la neurona a_1^1 recibe una señal de cada neurona de la capa anterior ($a_x^{(0)}$) a la que se le atribuye un peso ($w_{1,i}$), como se ve en la Figura 2.3, dicha neurona suma todos los pesos recibidos siendo la función σ la que comprueba si la neurona pasará su señal a la próxima capa.

La nomenclatura correspondiente a la Figura 2.3 es la siguiente:

- $a_x^{(c)}$: se refiere a una neurona como unidad matemática donde x es un indicador y c se refiere al número de capa.
- $\mathbf{a}^{(c)}$: se refiere al vector columna que aloja todas las neuronas de una capa c .
- $w_{i,j}$: se refiere al peso individual asociado a la señal mandada por la neurona j de la capa actual a la neurona i de la capa siguiente.

- $\mathbf{W}^{(c)}$: se refiere a la matriz de pesos que va desde la capa c hasta la capa siguiente.
- σ : hace alusión a la función de activación de la neurona $a_x^{(c)}$.
- $b_x^{(c)}$: es un factor extra que se introduce en el sumatorio de la neurona como *intercept*.
- $\mathbf{b}^{(c)}$: recoge en un vector columna todos aquellos valores *intercept* de una capa c .

El proceso de ML consiste en iterar valores dentro de la matriz $\mathbf{W}^{(c)}$ para lograr reducir al máximo el error de la respuesta.

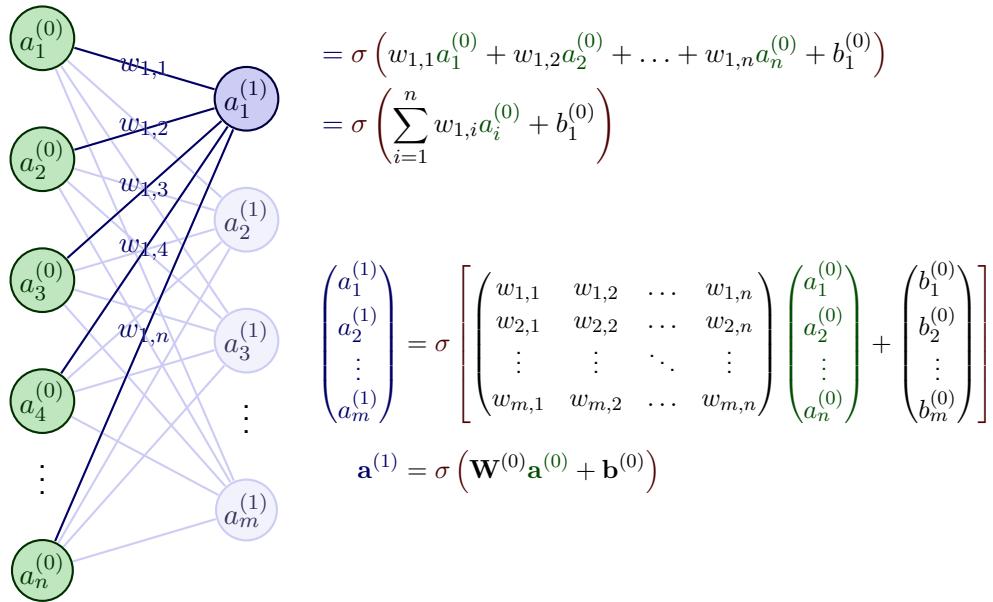


Figura 2.3: Expresión matemática que regula la activación de una neurona dentro de una red.

En la cuantificación del error se utilizan los siguientes indicadores:

- *Mean Squared Error* (MSE) o error cuadrático medio. Es el más usado y consiste en la media de todos los errores elevados al cuadrado. Este parámetro penaliza sobre todo los valores de error alto. El valor obtenido expresa el cuadrado de la unidad estudiada, por lo que, este valor no se puede trasladar al plano físico.
- MAE o error absoluto medio. Su valor numérico, como su propio nombre indica, es el error absoluto. Este error penaliza por igual todos los errores, por lo que los valores de error bajo, cobran mayor relevancia. Su cálculo es similar al MSE, utilizando el valor absoluto del error. Al tratarse de un error que se encuentra en la misma dimensión de la medida, permite su traslado al plano físico.
- RMSE o raíz del error cuadrático medio. La principal aplicación de este error consiste en penalizar los errores grandes, ya que se basa en valores de obtenidos al aplicar el cuadrado. La unidad de

medida de este error puede asimilarse con facilidad en el plano físico gracias a la raíz que reduce la dimensión que había quedado elevada al cuadrado.

- R^2 (R-squared). Este error se explica como la proporción de la varianza de los datos que es capaz de explicar el modelo. Puede tener valores que van desde 1 hasta valores negativos, siendo 1 que explica perfectamente la varianza de los datos, y siendo el valor negativo una predicción peor que el uso de media.

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (\mathbf{Y}_i - \hat{\mathbf{Y}}_i)^2 \quad (2.1)$$

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |\mathbf{Y}_i - \hat{\mathbf{Y}}_i| \quad (2.2)$$

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{n} \sum_{i=1}^n (\mathbf{Y}_i - \hat{\mathbf{Y}}_i)^2} \quad (2.3)$$

$$R^2 = 1 - \frac{\sum (\mathbf{Y}_i - \bar{\mathbf{Y}})^2}{\sum (\mathbf{Y}_i - \hat{\mathbf{Y}}_i)^2} \quad (2.4)$$

Donde:

- n : número de datos.
- \mathbf{Y}_i : valores reales.
- $\hat{\mathbf{Y}}_i$: valores predichos.
- $\bar{\mathbf{Y}}_i$ hace referencia a la media de los valores reales.

El modelo cinemático ha demostrado ser un método eficaz a la hora de obtener las etiquetas para el modelo de ML. Este procedimiento consiste en extraer las coordenadas del miembro o parte del cuerpo a estudiar, ya sea por métodos ópticos o bien inerciales.

Los sistemas ópticos con marcadores pasivos, obtienen las coordenadas a través de marcadores retroreflectivos situados en la zona objeto de estudio, de manera que, al grabar con un sistema de varias cámaras de infrarrojos alrededor de la muestra, su posición puede ser triangulada en el espacio, obteniéndose coordenadas de cada uno de los puntos marcados. Con este sistema se consigue un error submilimétrico [4], lo que lo convierte en un sistema ideal para la toma de modelos biomecánicos detallados y de forma fiable. Sin embargo, es necesario el uso de sistemas costosos y entornos controlados, que impiden el uso en paralelo de otros dispositivos (como el de toma de EMG). Por último, en el caso

aplicado de los dedos de la mano presenta desventajas debido al reducido tamaño de los marcadores necesarios, que puede provocar pérdidas o intercambios debido a occlusiones [4].

Una variación del anterior son los estudios realizados por Maggioni, Azevedo-Coste, Durand et al. [4], los cuales, llegaron a la conclusión que los sistemas ópticos sin marcadores, es decir aquellos cuyos datos de coordenadas eran extraídos de un modelo constituido por varios dispositivos de grabación simultánea, mediante el estudio del movimiento a través de un modelo esquelético de la mano por cinemática inversa del software *OpenSim*, mostraban un error cuadrático medio ligeramente inferior a los sistemas ópticos con marcadores basada en *Leap Motion Controller*.

En cambio, los sistemas inerciales portátiles, basados en acelerómetros triaxiales o giroscopios colocados sobre el cuerpo del paciente, arroja datos de aceleración lineal y velocidad angular en el punto marcado, permitiendo estimar la posición mediante un programa informático que extraiga las coordenadas [5]. La ventaja que tiene este sistema frente al óptico de marcadores pasivos, es que se trata de dispositivos de medida baratos, pequeños y portátiles, lo que ofrece muchas facilidades para la grabación (no es necesario un espacio limitado) lo que permite además hacer estudios continuos más extensos [5], [6]. Sin embargo, el inconveniente consiste en el efecto de deriva (*drift*) que aparece cuando ha transcurrido un periodo de tiempo largo desde su calibración. A lo largo del experimento y cuanto más tiempo pasa desde la última calibración se van introduciendo imprecisiones en las lecturas que provocan que la ubicación estimada se distancie cada vez más de la real. [6].

En la Tabla 2.2 se ha elaborado una tabla resumen con las ventajas y desventajas de cada técnica.

Tabla 2.2: Comparación entre las distintas tecnologías de captura de movimiento disponibles.

	Ventajas	Desventajas
Marcadores pasivos	Alta precisión Alta fidelidad	Alto coste Alta sensibilidad a occlusiones Requerimiento de vestimenta y espacio
Inerciales	Libertad de movimiento No hay occlusiones Bajo coste	Menor sensibilidad que ópticos Presencia de deriva (<i>drift</i>) Calibración necesaria
Visión artificial	Menor coste Constante mejora de precisión y fidelidad	Menor precisión (Mediapipe 10-40 mm [4])

Estudios recientes como los de Hioki y Kawasaki [7] han explorado cómo señales EMG de superficie pueden predecir continuamente movimientos y ángulos de los dedos usando algoritmos de aprendizaje automático, a través de un sistema de red neuronal recurrente con retardo temporal que estima los ángulos articulares de los dedos usando sólo 4 canales de sEMG en el antebrazo. Participaron 5 sujetos que realizaron dos patrones de movimiento dactilar, logrando estimar los ángulos con un error RMSE

de sólo 7.1–11.8 %, similar al de sistemas con 15 electrodos. Esto ilustra que incluso redes neuronales relativamente simples pueden aproximar las relaciones no lineales EMG–movimiento con datos limitados.

De forma similar, el equipo de Shrira, Reddy y Kosuri [8] emplearon comités de redes neuronales para decodificar el ángulo de la articulación de un dedo índice a partir de EMG; aunque su método logró detectar el patrón de flexo-extensión, reportaron dificultad en configurar un modelo robusto con pocos canales.

Con la disponibilidad de mayores volúmenes de datos y capacidad computacional, el aprendizaje profundo ha ampliado las posibilidades, así Ngeo, Tamei y Shibata [9] utilizaron datos de 10 sujetos que realizaron flexo-extensiones de dedos proponiendo un modelo de activación muscular con dos técnicas de regresión: una red neuronal *feedforward* y un regresor de proceso Gaussiano (GP). Reportaron coeficientes de correlación promedio de 0.85 (Articulación Metacarpofalángica (MCP)), 0.78 (Articulación Interfalángica Proximal (PIP)) y 0.73 (Articulación Interfalángica Distal (DIP)), con errores RMSE entre 5–15 %.

Lee, Kim y Park [10] entrenaron una red *encoder-decoder* con atención para estimar 14 ángulos articulares de la mano. Su modelo generalizó de movimientos de un solo dedo a gestos complejos. Avian, Prakosa, Faisal et al. [11] propusieron una red LSTM con atención y reducción de dimensionalidad mediante aprendizaje de variedades, logrando un coeficiente de determinación R^2 de 0.957.

Por su parte, Fan, Vargas, Kamper et al. [12] implantaron una red profunda multicapa para decodificar señales motoneuronales en EMG de alta densidad, logrando predicciones precisas de ángulos articulares y un control suave y estable.

Estudios como los que llevaron a cabo Ngeo, Tamei y Shibata [9] y Jiang, Muceli, Graimann et al. [13] usan captura de movimiento mediante sistemas ópticos con marcadores pasivos para obtener un modelo cinemáticos de la mano. Este es un procedimiento que requiere equipo especializado y que introduce un elevado coste al proyecto.

En cambio, otros trabajos sí incorporan visión artificial como fuente de referencia o apoyo. Stapornchaisit, Kim, Takagi et al. [14] combinaron EMG con un sistema de cámara CPM (Convolutional Pose Machine) y análisis ICA (Independent Component Analysis), mejorando las predicciones de ángulos respecto al uso exclusivo de EMG.

En definitiva, los modelos de aprendizaje profundo (*deep learning*) permiten predecir con alta precisión la cinemática de los dedos a partir de EMG superficial. El uso complementario de captura de movimiento da robustez al sistema, sin embargo, también es viable el uso de visión artificial para obtener un resultado similar. Estos enfoques suponen una base sólida para el desarrollo de prótesis activas y sistemas

interactivos basados en señales musculares.

3 Metodología y herramientas

En este proyecto se han empleado la siguiente serie de herramientas generales cuyas referencias se encuentran en el apartado anterior, por lo que únicamente se citan de manera sucinta.

- EMG superficial por electrodos, registrando datos de entrada a partir de las señales recogidas en modelos de brazo humano con una persona voluntaria. El equipo utilizado es un electromiógrafo de la marca BrainVision que se describe en una sección posterior de este capítulo.
- Se registraron 50 procesos de apertura y cierre de las articulaciones dando origen a un set de datos de 30.000 filas. Se utilizó como dispositivo de grabación una cámara Kinect de Xbox 360. Se realizaron dos horas de grabación totales que luego fueron procesadas en coordenadas y señal de EMG.
- Como modelo cinemático se eligió un modelo basado en visión artificial sin marcadores, y la herramienta Mediapipe desarrollada por Google que actúa como referencia objetiva de la postura de la mano, reduciendo así la ambigüedad de las etiquetas en el entrenamiento del modelo.
- Se desarrollaron por ML una red CNN y una red ANN.
- El entrenamiento fue a través de aprendizaje supervisado, donde las etiquetas fueron las coordenadas de la mano y los datos sobre los que predecir fue la señal de EMG.
- Se registraron los ángulos de las articulaciones de la mano predichas y las reales.
- Para valorar la eficacia del modelo se usó MAE, en conjunto con RMSE.

A continuación se explican aquellas herramientas más específicas que no han sido mencionadas anteriormente.

Para capturar la EMG se ha empleado el amplificador de instrumentación BrainAmp de la empresa BrainVision [15]. Este aparato está ideado para la captura de electroencefalografía. Sin embargo, en la propia página del producto se especifica que también es adecuado su uso para la captura de EMG.

Las especificaciones del dispositivo son: 32 canales de entrada más 2 referencias y tierras; frecuencia de muestreo de hasta 5 KHz; resolución mínima de $0.1 \mu\text{V}$ y un rango máximo de $\pm 3.28 \text{ mV}$.

Para la captura de potenciales en el brazo se usan unos electrodos adhesivos desechables de cobre, recubiertos con una fina capa de estaño y con gel para aumentar la conductividad. Estos electrodos son de la empresa Technomed [16].

Debido a que la captura de imagen y de EMG sucede en dispositivos distintos, será necesario introducir una señal dentro del amplificador para poder sincronizar ambos datos. Para esto se utiliza la solución comercial de BrainVision que permite introducir este tipo de señales en el amplificador para que lleguen hasta el *software* de lectura. El USB 2 Adapter [17] es el producto indicado para trabajar con nuestro amplificador de instrumentación. Como nexo entre ambos dispositivos se usa un Arduino [18] que recibe órdenes a través del puerto serie y convierte en señal eléctrica que se introduce en el USB 2 Adapter.

El *software* empleado para controlar este conjunto de aparatos ha sido BrainVision Recorder [19], distribuido por la misma empresa. Este programa fue ideado para recibir señales del amplificador y controlarlo. Se usa para grabar la señal obtenida y almacenarla para después procesarla. Una guía del procedimiento llevado a cabo con esta herramienta se puede ver en el Apéndice B.

Una vez obtenida la señal de EMG del anterior programa se introdujo en BrainVision Analyzer [20], el *software* diseñado para el filtrado y análisis de señales provenientes de electroencefalografía. De forma análoga a lo anterior, aunque se haya ideado con esta dirección, se pudo emplear para nuestro estudio ya que los filtros empleados son generales en cualquier análisis de señal. Entre los filtros usados en este proyecto se encuentran:

- Segmentación por señales: relacionado con la señal de inicio del vídeo y de final, fue empleado para sincronizar el vídeo con la señal EMG.
- FT: se usó para estudiar las frecuencias de los impulsos y la amplitud de los mismos.
- Remuestreo de la señal: usamos este filtro para relacionar la frecuencia de muestreo con la frecuencia de la captura de vídeo y conseguir una relación de número entero entre ambos números.
- Filtro *Notch*: usado para eliminar la influencia de la red eléctrica en la medida.
- Binarización: debido al funcionamiento del aparato obtenemos una medida de cada electrodo (habitual en ECG) y queremos simular una medida bipolar (habitual en EMG), por lo que se aplicó este filtro para restar canales seleccionados y obtener esta medida.

Un desarrollo más detallado del procedimiento seguido y los filtros aplicados se puede ver en el Apéndice C.

3.1 Procesado de la base de datos

Una de las partes más importantes del proyecto fue la del procesado de la base de datos, para esta tarea se empleó principalmente Python.

Python es un lenguaje de programación de alto nivel y sencillo muy utilizado en ámbitos de análisis de datos y ML. Esta herramienta fue útil para importar los datos de los vídeos y EMG, tratarlos y concatenarlos. Además se aprovechó el Interfaz de Programación de Aplicaciones (API) de tensorflow y se introdujó la base de datos generada en modelos de red neuronal. Para el procesado anterior se empleó la librería para el manejo de tablas de datos (*dataframes*), pandas, junto a la librería para procesos matemático *numpy* que sirvió a su vez para importar los datos de la EMG. Para el procesado de la imagen y obtención de coordenadas del modelo cinemático se usaron en conjunto las librerías *open-cv* y *mediapipe* respectivamente. El modelo cinemático extraído se representa en la Figura 3.1.

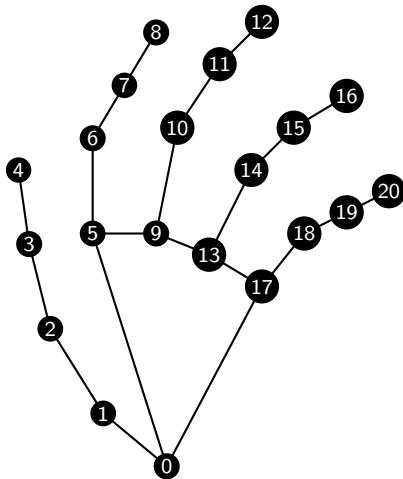


Figura 3.1: Esqueleto de la mano obtenido mediante la librería mediapipe con el número de articulación.

Durante el proyecto se ejecutó la mayoría del código generado de forma local. Sin embargo, al momento de entrenar modelos de ML se optó por usar Google Colab. Este servicio en la nube de Google permite escribir y correr *scripts* de Python en forma de *notebooks* en máquinas remotas con muchos más recursos.

Para apoyar todas estas tareas se empleó Conda [21], el servicio de creación y mantenimiento de entornos virtuales. De esta forma se consiguió una versión estable de las dependencias usadas otorgando mayor reproducibilidad al estudio. Conda es el módulo más simple del sistema de Anaconda, esto se puede ver en la infografía de la Figura 3.2. En el proyecto se ha necesitado debido a incompatibilidades para compilar la librería *libfreenect* entre *cython*, la librería para compilación de código en C a Python; y la versión actual de Python. Se necesitó retroceder la versión 3.10.16 de Python para encontrar

compatibilidad y poder así obtener las dependencias de `libfreenect` que permiten el control del Kinect con Python.

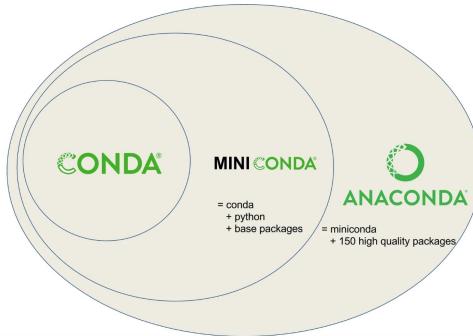


Figura 3.2: Infografía de la relación entre Conda, Miniconda y Anaconda como se ilustra en [22].

Finalmente, el código del proyecto se encuentra disponible en el repositorio de Github¹. Este servicio de almacenamiento en la nube se basa en el sistema de control de versiones de Git y permitió guardar versiones a lo largo de todo el proyecto. La estructura del repositorio se organiza en diferentes carpetas según a lo que estén dedicadas. Las principales tres carpetas están dedicadas a aquellos scripts enfocados en la obtención de imágenes, aquellos dirigidos al procesamiento de dichas imágenes y la base de datos y una última en la que se almacena lo relevante con el modelo de ML. También se deja una carpeta reservada para esta memoria.

3.2 Desarrollo metodológico

El desarrollo del trabajo se dividió en tres etapas. Una primera de captura de imagen y EMG en la que se sincronizaron ambos registros la toma de ambos datos, posteriormente se procesaron los datos obtenidos y se unieron en una misma base de datos. La última etapa consistió en el entrenamiento y validación de un modelo de ML.

Siguiendo la estructura del esquema presentado en la Figura 3.3, se puede ver que el proyecto empieza con una señal de *trigger* que marca el punto de inicio de grabación de vídeo y EMG. Una vez obtenidas las imágenes de vídeo, se llevó a cabo un procesado para extraer un esqueleto cinemático de la mano junto con sus coordenadas que luego se convirtió en ángulos de las articulaciones y se normalizó para todos los datos; por otro lado, se procesó la señal de EMG para eliminar la influencia de la red y adaptarla a la frecuencia de captura de vídeo.

A continuación, aunque en el esquema no se representa la unión de ambas rutas, en la práctica se

¹<https://github.com/CarMarFu/PFG>

generó un set de datos con todo lo anterior y se entrenó un algoritmo con ello. Este algoritmo de ML se entrenó de tal forma que su entrada era el fragmento de impulsos eléctricos que le corresponden a un fotograma y la etiqueta era el ángulo que adquirió el dedo en ese mismo fotograma. También se probaron otros tipos de entrada y salida, como dar información del ángulo anterior o predecir únicamente la diferencia de ángulo con el *frame* anterior.

Con este modelo entrenado, se trató de verificar su calidad mediante análisis de parámetros y la representación de valores predichos junto a reales.

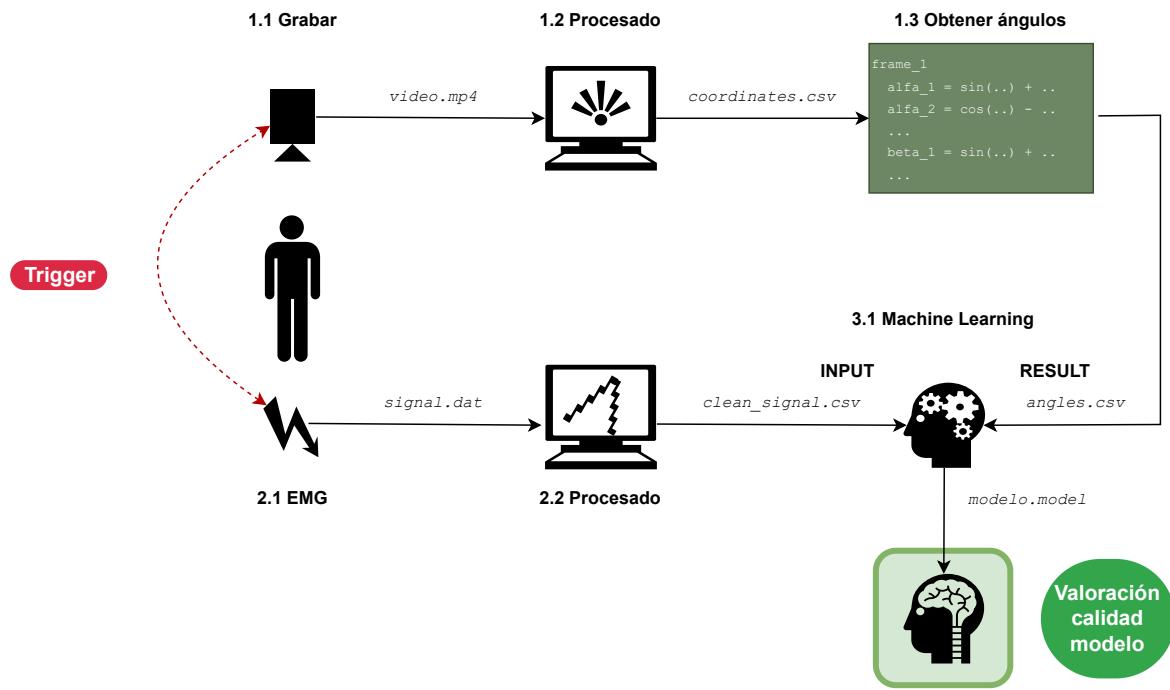


Figura 3.3: Esquema del desarrollo metodológico seguido en este PFG

4 Desarrollo

4.1 Estudio previo y calibración señales

En la siguiente sección se documenta el proceso previo de calibración y análisis de señal en preparación de la captura de imágenes. De esta forma se buscó justificar las decisiones tomadas en lo referente al protocolo de toma de imágenes.

4.1.1 Reconocimiento de imagen

Al emplear reconocimiento de imagen por IA es necesario que los resultados obtenidos sean lo más robustos posible. De esta forma se estudió la posición más óptima de la mano delante de la cámara para maximizar la confianza de la mano y más tarde se analizó la estabilidad del modelo cinemático extraído, buscando minimizar la vibración debido a obstrucción de la visión de marcas importantes. Para ello se estudió la mano en diferentes posturas y con diferentes complementos.

Confianza del modelo y mejor postura

Se preparó un *script* que muestre en directo el *frame* procesado con el modelo dibujado junto a la confianza. Se probó a situar la mano de cinco formas diferentes, con la palma o el dorso mirando hacia la cámara, y cada una de ellas se probó con la mano paralela al plano de la lente y con una ligera supinación y, en el caso de la palma mirando hacia el objetivo, también se probó con una leve pronación.

Se descartó el caso de la mano pronada cuando el dorso está enfocado hacia el objetivo debido a que se ocultan las marcas del pulgar al cerrar el puño. A pesar de que la confianza fue alta, se detectó mucho ruido en las lecturas.

También se añadió al estudio la mano cerrada y sujetando una pinza de entrenamiento. Los resultados

para la palma se pueden ver en la Figura 4.1 y para el dorso se pueden ver en la Figura 4.2.

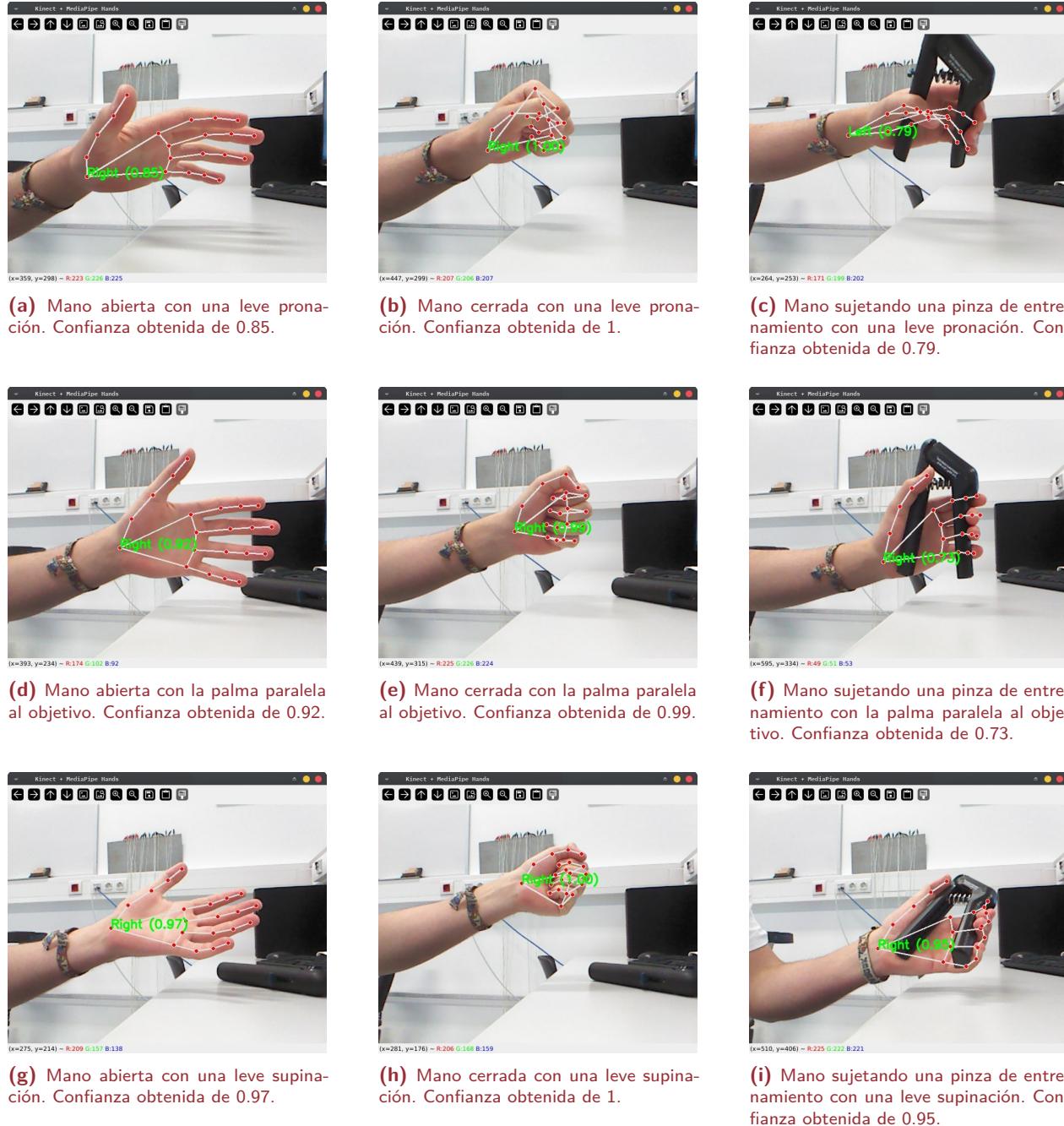


Figura 4.1: Capturas de la confianza para encontrar la mano (*tracking confidence*) del modelo cuando la palma de la mano se encuentra mirando hacia el objetivo.



Figura 4.2: Capturas de la confianza para encontrar la mano (*tracking confidence*) del modelo cuando el dorso de la mano se encuentra mirando hacia el objetivo.

Al añadir la pinza de ejercicio se perdió mucha información en las manos. La ocultación de puntos clave para el análisis de la mano perjudicó al modelo cinemático extraído. Los efectos de esta incertidumbre se ven claramente en la Figura 4.2f, en la que se perdió información de la punta de los dedos y acabó posicionando la marca del pulgar en un lugar que no correspondía al real.

De entre todas esas muestras, las posiciones más claras fueron las que situaban la palma hacia la cámara o con una ligera supinación. La correcta visión de los dedos (en especial del pulgar) es crucial para la detección de un buen modelo cinemático. Aquellas posiciones con el dorso hacia la cámara provocaban que el pulgar no se pudiera encontrar en absoluto.

En resumen, los registros de los movimientos de la mano, que se obtuvieron de las posiciones representadas en las Figuras 4.1 y 4.2, fueron aquellos que presentaron valores óptimos de Confianza (0,99-1). Estos fueron aquellos que tenían la palma en dirección a la cámara y perpendicular o con una ligera supinación.

Estabilidad de la lectura

Al obtener las imágenes anteriores se observó una vibración en la posición de los puntos clave al aplicar el modelo. Se grafica la coordenada x de los dedos para analizar la anterior vibración en las diferentes posturas. En la Figura 4.3 se muestra lo obtenido al abrir y cerrar la mano con la palma mirando hacia el objetivo de la cámara, se descartaron los experimentos con el dorso hacia el objetivo ya que al perder de vista la punta de los dedos no se obtiene el rigor deseado en la medida.

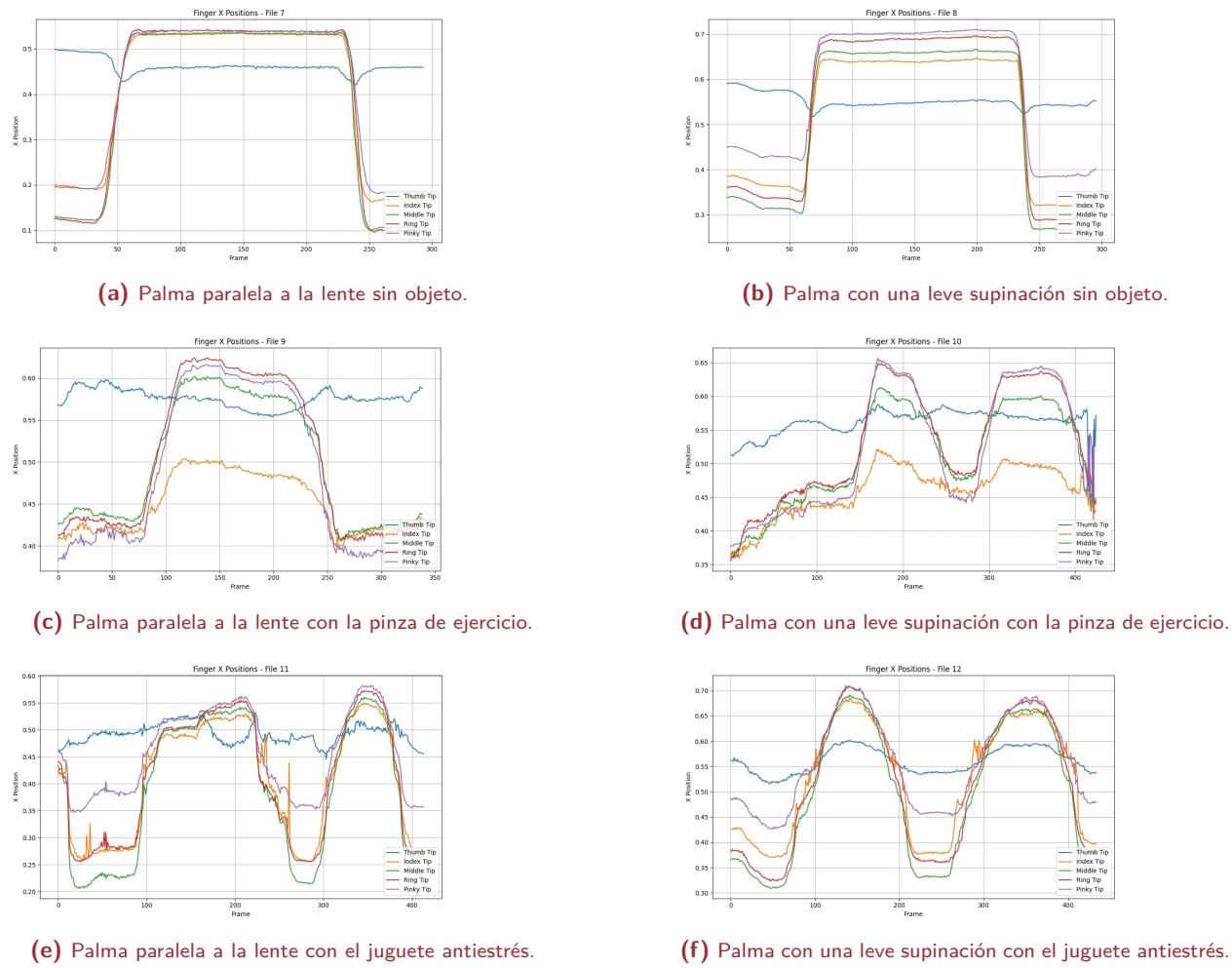


Figura 4.3: Posición en la coordenada x de los dedos a lo largo del tiempo al abrir y cerrar la mano.

Se puede observar en las Figuras 4.3a y 4.3b que cuando no se sostuvo nada en la mano se observa toda la mano con claridad y no se obtiene apenas ruido. También se vió que en comparación entre el juguete y la pinza, el juguete devolvió una lectura más clara.

Se puede concluir que para capturar el movimiento es mejor grabar la mano con una ligera supinación para dar más perspectiva sobre la flexión de los dedos como muestran las gráficas de los ángulos de los dedos, representados en las Figuras 4.3a y 4.3b. Para obtener los datos durante la contracción de la mano, se obtuvieron registros más limpios, cuando se sujetaba el juguete que cuando se sujetaba la pinza.

Suavizado de la posición y ruido

Aún con la mejor posición, se detectó ruido dentro del set de datos debido a imprecisiones a la hora de colocar las marcas de puntos no visibles de la mano, provocando que los puntos parpadeen en varias localizaciones cercanas al punto real hasta que se vuelve a tener visibilidad plena de la sección. Se estudió la posibilidad de realizar un filtro para suavizar las señales.

El filtro seleccionado fue un filtro de Savitzky-Golay que se ajustó para conseguir la señal más suave sin perder información. Este filtro es ideal al consistir en un impulso variable que afecta más a aquellas regiones con mayor ruido, dejando los puntos del set que sí han conseguido una visibilidad correcta sin alterar. Este filtro ya se ha usado para señales como las obtenidas de un electrocardiograma [23], por lo que nos fue útil para el movimiento de los dedos.

Se usó la implementación del filtro de la librería `scipy.signal` en Python que nos ofreció dos parámetros para ajustar:

- **Window:** cantidad de datos tomados para suavizar. Valor impar que define los datos introducidos en la función cuadrática para suavizar el dato. Esta cantidad de datos tomados se encuentra centrada en el valor actual, es decir, si se cogen 33 valores, significa que se usarán 16 valores previos y otros 16 posteriores para el computo del valor. Se buscó que fuera un valor alto ya que la señal de movimiento de una mano es suave y continua. No debería presentar picos a menos que se padeciera alguna enfermedad degenerativa que provoque el temblor de la mano. Por lo que se escogió una ventana de 33 datos ya que la grabación fue a 32 Fotogramas por Segundo (FPS).
- **Order:** orden del polinomio usado para la aproximación. De nuevo, las relaciones entre los datos de una señal proveniente del movimiento de una mano son lineales y sencillos, por lo que se empleó un polinomio de segundo grado para poder filtrar bien las curvas que sí que se presentan en los datos.

A continuación, en la Figura 4.4, se muestra una comparativa de la señal cruda y suavizada con el filtro propuesto. Se presenta la coordenada con mayor ruido del set. Esta clasificación de ruido se obtuvo siguiendo el indicador obtenido de la media del valor absoluto de la primera derivada de la señal, es decir, la media de la pendiente a lo largo de toda la señal.

$$\text{Ruido} = \frac{1}{n} \sum_{t=1}^{t=1} |f(t+1) - f(t)| \quad (4.1)$$

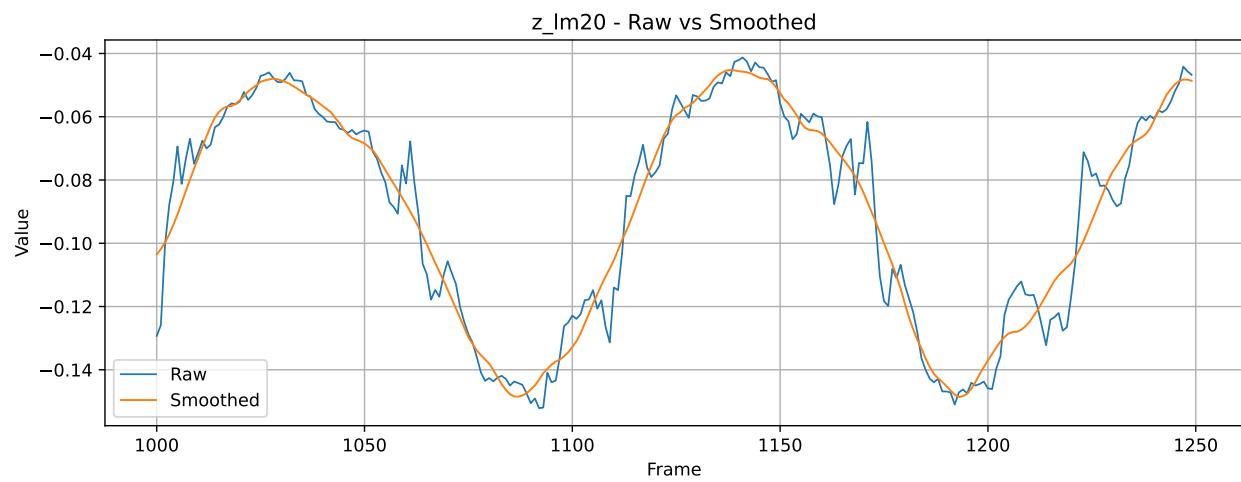


Figura 4.4: Comparación de la coordenada z de la punta del meñique antes y después de usar el filtro de Savitzky-Golay

4.1.2 Señales de EMG

La posición del brazo afectó a la lectura de la EMG [13]. Se necesitó hacer un estudio previo con el equipo para obtener conclusiones de cara a una toma de datos lo más limpia posible.

Como se indica en la introducción del PFG, el objetivo es el análisis de la flexión y extensión de todas las falanges de la mano. Por lo que se propusieron dos aproximaciones en cuanto a la ubicación de los electrodos. La primera, ubicar los electrodos buscando la señal de músculos concretos [9] relevantes para los movimientos estudiados. La segunda, colocar los electrodos en forma de pulsera [13] a una distancia establecida para captar la señal de varios músculos a la vez y explotar las capacidades del algoritmo de ML para detectar patrones.

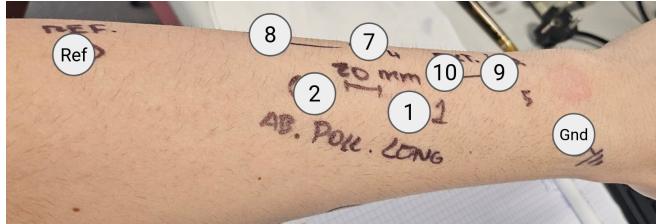
A continuación, en la Tabla 4.1 se recogen los músculos de interés para nuestro estudio junto al electrodo que se colocó [9].

Tabla 4.1: Músculos elegidos para el estudio y sus electrodos junto al dedo que afecta.

Electr.	Músculo	Dedo
1 y 2	Abductor largo del pulgar	Pulgar
3 y 4	Flexor superficial de los dedos	Flexión de la PIP desde el índice al meñique
5 y 6	Flexor profundo de los dedos	Flexión DIP desde el índice al meñique
7 y 8	Extensor común de los dedos	Dedos desde el índice al meñique
9 y 10	Extensor del índice	Índice

Por tanto, se colocaron un total de 10 electrodos, en los músculos del antebrazo implicados en los movimientos de los dedos. Para localizar el lugar preciso donde colocar cada electrodo del antebrazo, se realizaron movimientos de flexión y extensión de cada una de las falanges recogidas en la Tabla 4.1. De esta forma se puede detectar con facilidad la fibra que actúa. El resultado para el brazo izquierdo se puede ver en la Figura 4.6. Se decidió utilizar la mano derecha para el registro de los datos, puesto que el sujeto voluntario era diestro, y por tanto, los músculos a estudiar presentaban un mayor desarrollo que en el caso de la mano izquierda, y por tanto, mejores registros de EMG, es decir, se colocaron los electrodos de forma specular a lo mostrado en la anterior Figura 4.6.

Finalmente se optó por la disposición de los electrodos en forma de brazalete, como se observa en las fotografías de la Figura 4.7, utilizando el modelo más aproximado al prototipo que este PFG propone, ya que el brazalete, supone una mayor facilidad de auto-instalación al propio paciente.

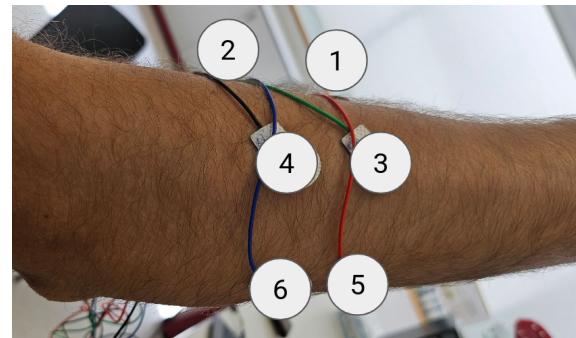


(a) Imagen del dorso del brazo.

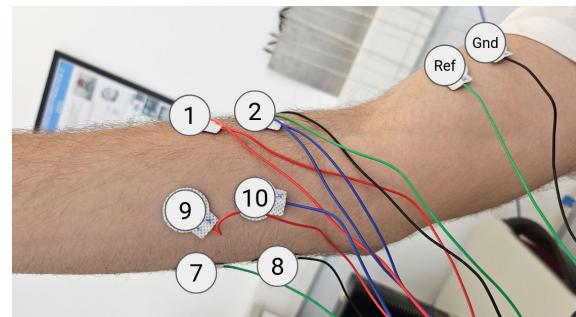


(b) Imagen del anverso del brazo.

Figura 4.6: Posición de los electrodos en la ubicación muscular en el brazo izquierdo de un voluntario.



(a) Imagen del dorso del brazo.



(b) Imagen del anverso del brazo.

Figura 4.7: Posición de los electrodos en "pulsera" en el brazo derecho de un voluntario.

Setup

El *setup* experimental se muestra en la Figura 4.8. Se buscó que el hombro estuviese en reposo para evitar fatiga y que el brazo estuviera lo más relajado posible reduciendo el ruido introducido por una muñeca en tensión [13]. También se buscó una posición elevada por encima de la superficie para permitir abrir y cerrar la mano con facilidad, el peluche proporcionó un apoyo cómodo y óptimo para la captura de imagen.

Además se preparó un circuito con dos interruptores para introducir manualmente un *trigger* cuando se abrió y se cerró la mano, el circuito descrito se puede observar en la Figura 4.9.

Una vez colocados los electrodos y preparado el circuito se inició BrainVision Recorder y se siguió el procedimiento detallado en el Apéndice B.

Se consultaron las impedancias obtenidas y se obtuvo que, para los electrodos de referencia y tierra se consiguió una impedancia de $9 \text{ K}\Omega$ y mayor de $100 \text{ K}\Omega$ respectivamente. Como se mencionó en la metodología, el aparato está preparado para encefalografía, y marcó las impedancias de los electrodos

como inapropiadas. Esto fue debido a que la piel en el antebrazo es mucho más gruesa que en el cuero cabelludo, y en el caso de la cabeza, se usa un sistema compuesto por un casco y gel conductor. Debido a que el potencial eléctrico de las señales eléctricas de los músculos, son mayores que lo potenciales obtenidos del cerebro, no fue necesario gel conductor, obteniéndose una señal fácilmente procesable.

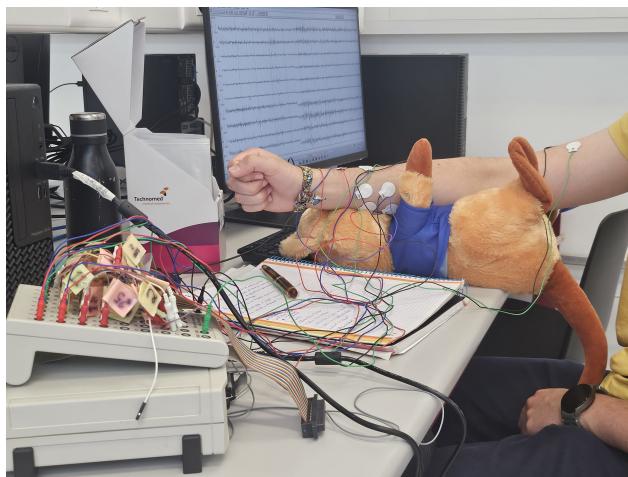
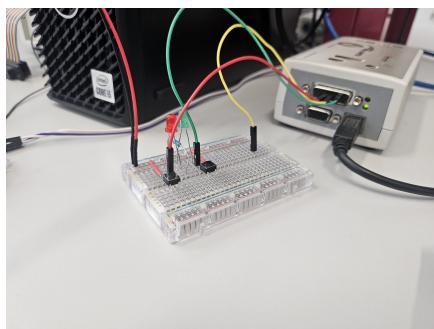
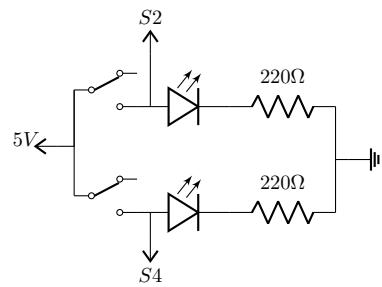


Figura 4.8: Imagen del *setup* experimental con los electrodos posicionados y el soporte para el brazo situado. A la izquierda se ve el amplificador de instrumentación conectado.



(a) Fotografía del sistema montado.



(b) Esquema de conexiones empleado.

Figura 4.9: Imagen del sistema de conexiones para mandar señales manuales durante el experimento.

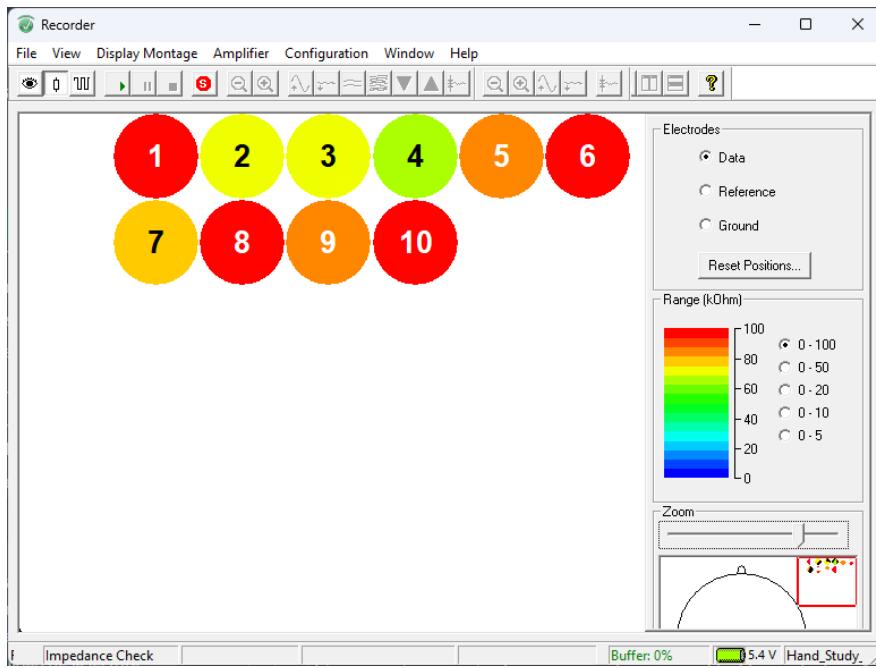


Figura 4.10: Captura de la pestaña de impedancias de BrainVision Recorder.

Se comprobó desde el visor la obtención de señal y la lectura de los *triggers*. Y se procedió a grabar la EMG realizando movimientos suaves y aplicando algo de tensión cuando se llegó al final del movimiento (apretando algo el puño o estirando algo la mano). Los *triggers* que se mandaron y que marcaban el comienzo y final del registro, se hicieron como se describe a continuación:

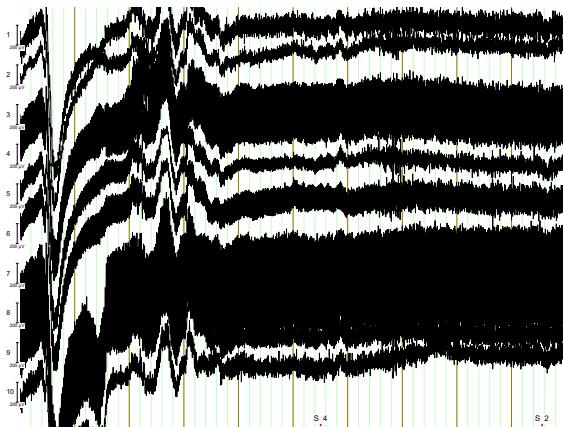
- Con la mano abierta y antes de empezar a cerrar el puño, se pulsó el botón con el cable rojo conectado (S4).
- Una vez cerrada la mano y antes de empezar a abrirla, se pulsó el botón con el cable verde conectado (S2).

Estudio comparativo de apertura y cierre de la mano

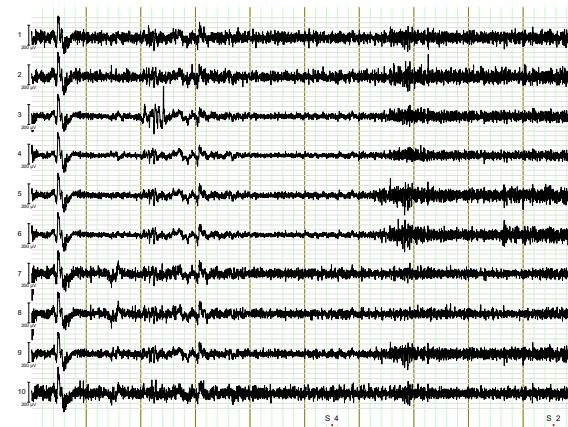
Para ello se llevó a cabo el procedimiento descrito en el Apéndice C donde se muestra el proceso completo que se ha llevado a cabo para la integración y procesado de las señales EMG y los datos obtenidos de los ángulos de movimiento de las falanges con BrainVision Analyzer y Python. Los filtros usados fueron los que se ven en la Tabla 4.2, y fueron análogos a lo que se hizo para Recorder en el apartado de visualización- Se consiguió lo que se ve en la Figura 4.11b.

Tabla 4.2: Parámetros para el filtrado de la señal cruda [13].

Filtro		Remuestreo
Pasa-banda	Notch	
10-450 Hz	50 Hz	1024 Hz



(a) Señal cruda obtenida del EMG.



(b) Señal obtenida después de aplicar los filtros al EMG.

Figura 4.11: Comparativa entre señal cruda y después del filtro de frecuencias que se ve en la Tabla 4.2.

Se continuó el procesado de la señal para segmentar según los marcadores mandados, de tal forma que se agruparon las señales provenientes de abrir o cerrar la mano. Se estudió en paralelo el resultado con todos los canales. También se fusionaron parejas de canales para llevar a cabo un análisis bipolar. Finalmente se hizo una FT con un promedio para facilitar la lectura.

Los resultados obtenidos con los electrodos en la posición sobre el músculo se presentan y se comparan en la Figura 4.12 en la que se presentan en un mismo gráfico la señal bipolar obtenida para el movimiento de apertura y de cierre de la mano. Se aporta también el *p.value* de un *paired t-test* entre la señal recibida durante los dos movimientos de la mano. De esta forma se obtuvo una conclusión sobre la diferencia estadística que presentaron los dos tipos de señales.

Se hizo lo mismo para las señales obtenidas del posicionamiento en forma de “pulsera” de los electrodos. Estos resultados se pueden ver en la Figura 4.13.

Tras el estudio de las gráficas se pudo deducir que sí hay una diferencia relevante entre las señales al abrir y cerrar la mano en ambos casos. Se compararon los valores de *p.value* en la Tabla 4.3. Estos valores se han ordenado de mayor a menor relevancia estadística. Además, en rojo se han marcado los *p.values* menos significativos y en azul los ceros obtenidos. Estos ceros se produjeron al obtener un *p.value* tan

bajo que excedía la precisión mínima de una variable de tipo coma flotante.

Con esto se pudo ver que se obtuvo un cero más al buscar la ubicación anatómica. Comparando ambos planteamientos de vió que eran igual de viables para realizar el estudio. Debido al enfoque de este estudio basado en ML, se terminó optando por el formato en forma de “pulsera” que aprovecha las virtudes del aprendizaje automático y ofrece la posibilidad de una implementación sencilla. Uno de los objetivos de este estudio es la posibilidad de ser implantado en una persona real, por lo que también se consideró más sencillo colocar una banda con los sensores integrados, que pedirle al paciente que ubique los electrodos buscando los músculos seleccionados.

Tabla 4.3: Comparativa de los *p.values* obtenidos en la Figura 4.12 y Figura 4.13 ordenados de mayor a menor relevancia estadística.

Ubicación muscular	“Pulsera”
0	0
0	2.328e-218
4.997e-295	3.407e-44
9.891e-38	1.249e-35
2.021e-04	8.669e-25

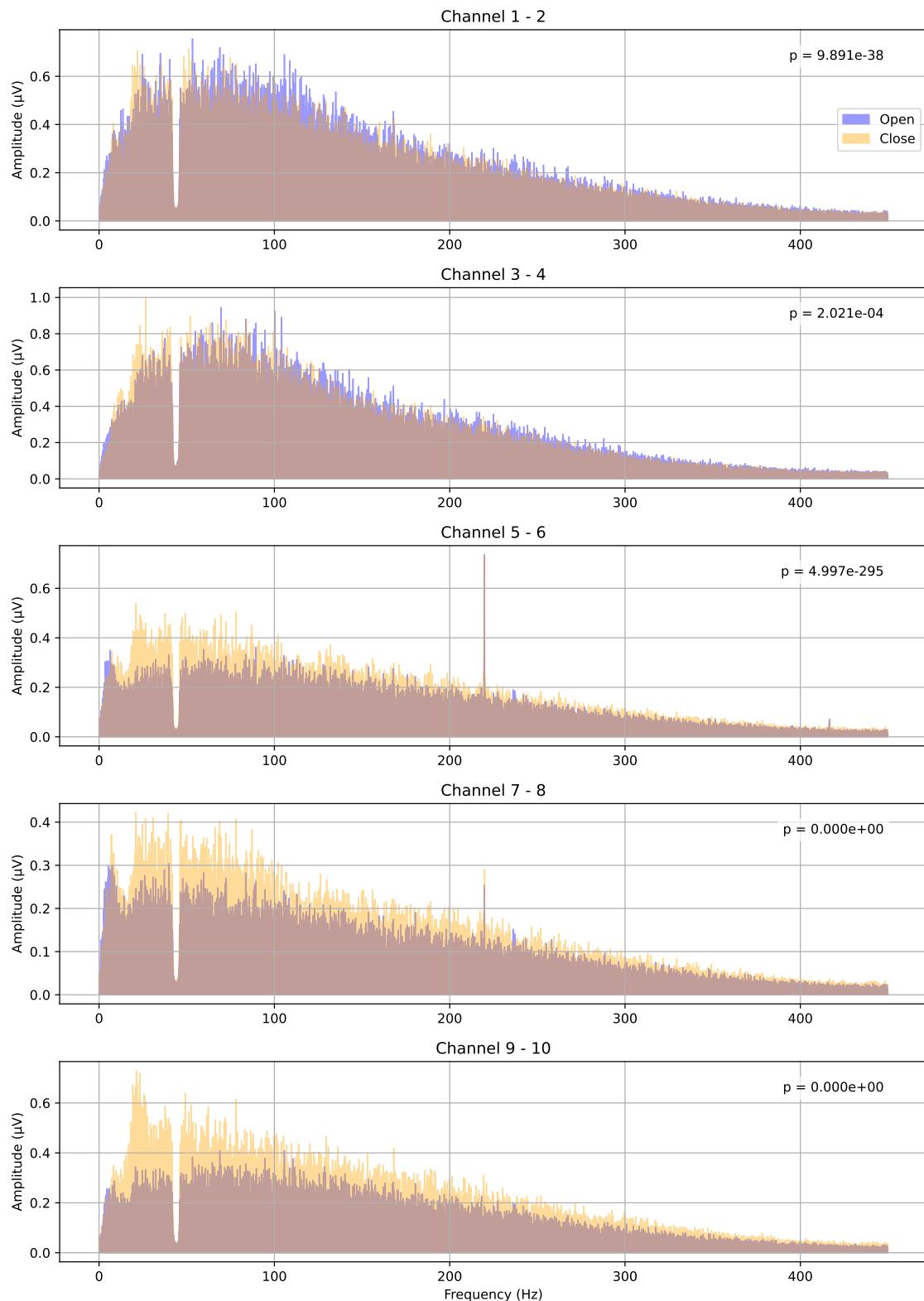


Figura 4.12: Comparativa de la FT entre la apertura y cierre de la mano para los pares de electrodos según la disposición muscular de la Figura 4.6.

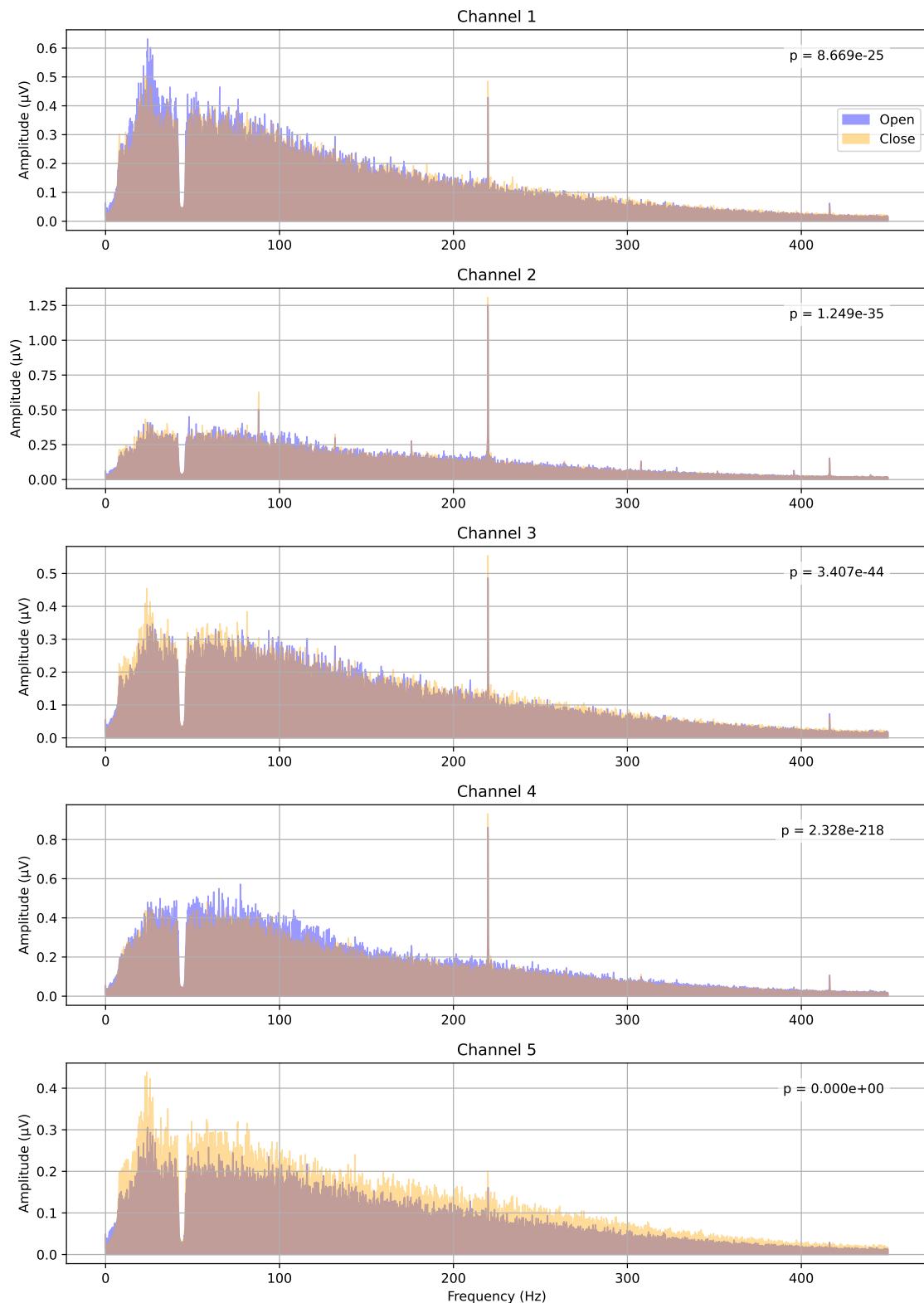


Figura 4.13: Comparativa de la FT entre la apertura y cierre de la mano para los pares de electrodos siguiendo la disposición de “pulsera” de la Figura 4.7.

Estudio del ruido de muñeca

Con la intención de preparar un filtro para eliminar el ruido introducido por la muñeca se analizaron las frecuencias predominantes al comparar la EMG con la mano relajada y la mano en la posición para la grabación de imagen. Siguiendo la conclusión obtenida en el apartado anterior, este análisis se llevará a cabo únicamente para cuando los electrodos se sitúan en forma de pulsera.

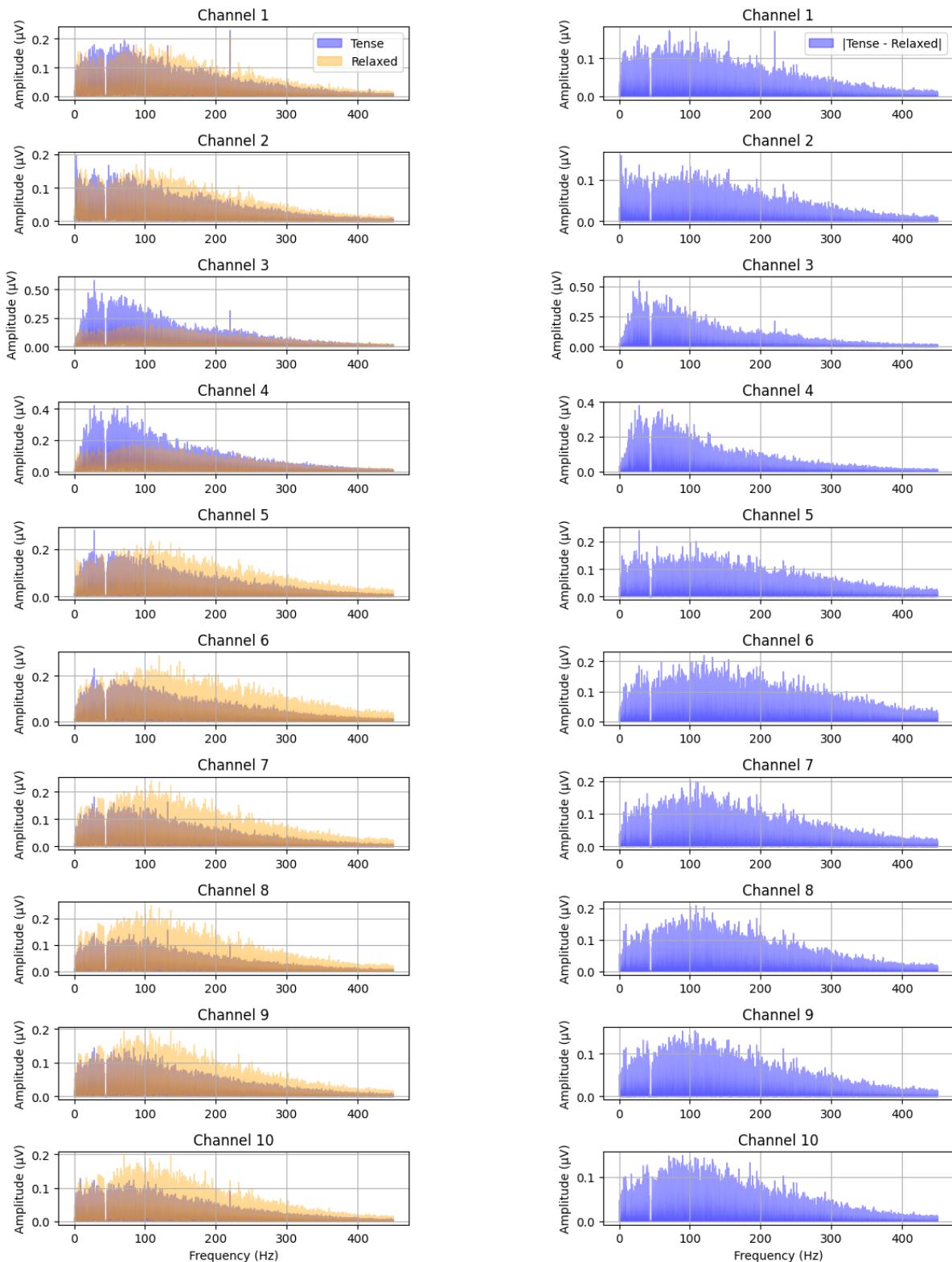
Para este análisis en lugar de unir los electrodos calculando la señal bipolar, se trabajó con los 10 canales por separado.

Se llevó a cabo la comparación de los espectros de frecuencias obtenidos mediante una FT. En la Figura 4.14a se presentan las amplitudes detectadas una contra la otra, y en la Figura 4.14b se presenta la diferencia en valor absoluto de lo que se presenta en la Figura 4.14a.

De la Figura 4.14 se extrajo que los canales más afectados fueron el 3 y el 4, esto es lógico debido a que bajo esos electrodos se encuentran los músculos que se encargan de levantar la muñeca. También se pudo ver como en el resto de canales (sobretodo del 5 al 10) la amplitud de la señal fue mayor al tener la mano relajada, esto fue debido a que estos músculos se comprimen al relajar (dejar caer) la mano y se deben contraer, en menor medida que al ejercer fuerza.

Al analizar la diferencia en valor absoluto (Figura 4.14b) se pudo ver en las escalas de amplitud que, al igual que antes, que los canales 3 y 4 son los más afectados debido a la altura de su pico más alto.

Finalmente, se trató de encontrar el rango de frecuencias que más ruido presentó, con la intención de elaborar un filtro que eliminase o mitigase el ruido. Sin embargo, no se logró separar esa frecuencia del resto, y se terminó optando por conservar la señal integra a fin de no perder información de cara al aprendizaje automático.



(a) Comparación de la amplitud de la señal en frecuencia con la muñeca tensa y con la muñeca relajada.

(b) Valor absoluto de la diferencia de amplitudes en frecuencia.

Figura 4.14: Estudio del ruido introducido en cada canal a causa de una muñeca en tensión.

4.1.3 Setup para grabación de imagen y EMG simultánea

En esta sección se describe el procedimiento para conectar la cámara, el Arduino y el amplificador de instrumentación de la EMG seguido. Debido al *software* empleado para la captura de señal de EMG y su disponibilidad exclusiva en uno de los ordenadores de la escuela, el estudio se lleva a cabo desde dos ordenadores diferentes. Uno para la señal y otro para la imagen.

Como se ha mencionado anteriormente, el objetivo del proyecto fue predecir coordenadas extraídas del vídeo y relacionarlo con señales de EMG, por lo que se buscó una relación de número entero entre ambos datos. Se tomó la decisión de grabar la imagen a 32 FPS y de muestrear la señal a 1024 hercios, consiguiendo (idealmente) una relación de 1 a 32 *frame* a impulsos eléctricos.

Captura de imagen

Para la captura de la imagen se preparó el siguiente Algoritmo 1 en Python con la ayuda de las siguientes librerías:

- serial
- freenect
- cv2
- os

Con este *script* lo que se buscó fue guardar el vídeo captado por la Kinect y enviar una señal a través del Arduino para lograr sincronizar el vídeo con la EMG, o lo que es lo mismo, las coordenadas obtenidas a través del vídeo con los impulsos eléctricos de la señal de EMG. Dentro del algoritmo, sólo el bucle `while` guarda las imágenes (líneas 7 a 10 en Algoritmo 1), pero debido a un desfase entre el envío de la señal de comienzo y el verdadero comienzo del vídeo, fue necesario hacer una inicialización previa del bus de datos con la Kinect. Es por esta razón que se crea la ventana con un *frame* arbitrario antes de empezar a grabar el vídeo (líneas 4 y 5 en Algoritmo 1).

Este *script* se estructuró de forma que primero inicializan varias herramientas como la conexión por puerto serie con Arduino, la variable que va a alojar nuestros *frames* (se formatea para que aloje un vídeo en códec mp4v a 32 FPS con una calidad de 640x480 píxeles de definición) y la ventana en la que se va a mostrar la imagen que está siendo obtenida del Kinect. Con una función propia también se busca dentro de la carpeta de destino un nombre para el vídeo que saldrá como resultado.

Una vez todo queda iniciado, se manda una señal por puerto serie al Arduino que reenvia a través de un cable al BrainVision USB 2 Adapter (*trigger S1* de inicio). De aquí se inicia un bucle que capturará imágenes desde nuestro Kinect y las almacenará en la variable de vídeo anterior y que durará hasta que el usuario presione la tecla “Q” en el dispositivo de captura de imagen.

Una vez finalizado el experimento (cuando el usuario presione la tecla especificada), se envía una señal al Arduino de nuevo para indicar el final de la grabación (*trigger S4* de final) y se guardan los *frames* obtenidos como un vídeo. También se cierra la conexión con el Arduino y la ventana en la que se podía ver el *frame* capturado en tiempo real.

El *script* de Arduino (Algoritmo 2) consiste simplemente en inicializar los pines de salida 8 y 9 junto a la comunicación por puerto serie (*setup*). Luego en el bucle principal (*loop*) se preparan una serie de condiciones que se verifican cada vez que el Arduino está listo para leer el puerto serie. Con una condición *if* se comprueba si el comando es una “H” o una “L” para encender y apagar consecutivamente el pin 8 o 9 y marcar el inicio o final de la grabación respectivamente.

Algoritmo 1: Grabación de vídeo con Kinect y envío de señal por Arduino.
Database_Processing/record.py

Data: Imagen vía Kinect
Result: Vídeo grabado y señales de Arduino

```

1 Inicializar conexión serial con Arduino ;                                // /dev/ttyACM0, 9600 baudios
2 output_filename ← GetNextFilename() ;                                       // Generar nombre de archivo
3 Crear objeto output_video ;                                              // CÓDIGO mp4v, 32 fps, resolución 640x480
4 frame ← GetVideo() ;                                                     // Iniciar cámara y ventana
5 Mostrar ventana "Kinect Video" con el frame;
6 Enviar H al Arduino ;                                                    // Trigger de inicio S1
7 while usuario no pulsa q do
8   frame ← GetVideo();
9   Escribir frame en output_video;
10  Mostrar frame en ventana "Kinect Video";
11 Enviar L al Arduino ;                                                   // Trigger de fin S4
12 Liberar output_video;
13 Cerrar ventanas de OpenCV y conexión con Arduino;
```

Captura de EMG

La captura de EMG es análoga a como se ha llevó a cabo en anteriores secciones, con el matiz de que en este punto solo se recibió una señal de inicio (S1) y otra de final (S2). Por lo que al momento de la segmentación con BrainVision Analyzer (después de la grabación y el guardado con BrainVision como se explica en el Apéndice B) solo se obtuvo un gran segmento. El proceso completo llevado a cabo para el procesado de la señal de EMG se detalla mejor en el Apéndice C.

Algoritmo 2: script para recibir órdenes del Algoritmo 1 en el Arduino.
Database_Processing/trigger_receiver/trigger_receiver.ino

Data: Comandos vía puerto serie
Result: Activación de pines 8 y 9 según comando recibido

```

1 Configurar pin 8 y 9 como salida;
2 Iniciar comunicación serie a 9600 baudios;
3 while el Arduino esté encendido do
4   if hay datos disponibles en el puerto serie then
5     Leer carácter recibido y almacenarlo;
6     if 'H' then
7       Activar pin 8;
8       Esperar 100 ms;
9       Desactivar pin 8;
10    else if 'L' then
11      Activar pin 9;
12      Esperar 100 ms;
13      Desactivar pin 9;

```

Calibración de la relación fotograma/impulso

Aunque se tomaron todas las precauciones en el Algoritmo 1 para que no hubiera ningún tipo de desfase entre la señal de Arduino y la captura de la cámara. Tras comparar la longitud del vídeo (en FPS) y de la señal de EMG se encontró que la relación es cercana a 32 (32,04), pero no exacta. Esto pudo ser debido a la capacidad de procesamiento del ordenador¹ usado para captar las imágenes. Por esto, fue necesario un preprocesado de los datos para eliminar aquellos impulsos sobrantes. Por como se planteó el Algoritmo 1, el desfase fue introducido en las líneas 9 a 11 debido a que la señal de salida no se manda hasta después de haber procesado la imagen para mostrarla, por lo que en el siguiente paso, durante la confección del set de datos, se eliminaron los impulsos truncando la señal en la parte final.

4.2 Confección del set de datos

Una vez obtenido el vídeo del movimiento de la mano y la señal de EMG por separado, se describe el proceso empleado para unir todo en un mismo set de datos. El proceso seguido se puede ver representado de forma general en el Algoritmo 3, pero se describe en detalle a lo largo de toda esta sección.

¹GPD P2 Max con Arch Linux (x86_64), CPU Intel Core m3-8100Y (4) @ 3.40 GHz, 15,9 GiB de RAM y GPU integrada Intel UHD Graphics 615

Algoritmo 3: Pipeline seguido para la confección del set de datos.

Data: Vídeo del movimiento de la mano y señal EMG
Result: Set de datos unido

- 1 Importar datos de vídeo;
- 2 Importar datos de EMG;
- 3 Inicializar pandas.DataFrame que alojará el set de datos;
- 4 **for** cada frame del vídeo **do**
- 5 Extraer modelo cinemático;
- 6 **for** cada landmark del modelo cinemático **do**
- 7 Guardar coordenadas en el DataFrame;
- 8 Suavizar coordenadas;
- 9 Calcular ángulos de flexión;
- 10 Formatear cada fila un frame en el DataFrame;
- 11 Truncar señal EMG;
- 12 **for** cada fila del DataFrame **do**
- 13 **for** cada canal del EMG **do**
- 14 Añadir 32 siguientes impulsos;

4.2.1 Procesado de vídeo

La parte del proceso que a continuación se describe está englobada en las líneas 1 y 4-10 del Algoritmo 3.

Se empezó importando el vídeo con la ayuda de la librería open-cv junto con el modelo preentrenado de Google [24] para el reconocimiento de manos. Luego se definieron los parámetros del modelo para introducir un vídeo y que reconozca como máximo una mano en escena como se puede ver en el Listado 1.

Python

```

1 base_options = python.BaseOptions(model_asset_path=MODEL_PATH)
2 options = vision.HandLandmarkerOptions(
3     base_options=base_options,
4     running_mode=vision.RunningMode.VIDEO,
5     num_hands=1
6 )
7 landmarker = vision.HandLandmarker.create_from_options(options)

```

Listing 1: Opciones para definir el modelo de reconocimiento de mano por visión artificial de Mediapipe

Después de esto con un bucle se recorrieron todos los *frames* del vídeo aplicando el modelo y guardando en un DataFrame de pandas las coordenadas obtenidas. A la vez se guardó también en un fichero .csv. La estructura seguida para almacenar estos datos fue la siguiente: cada fila corresponde a un *landmark* de un *frame* concreto.

Además se generó un vídeo en el que superpuso el esqueleto inferido sobre lo grabado que sirvió como comprobación del correcto y completo análisis del vídeo.

Se buscó que los datos tuvieran una estructura de tal forma que cada fila correspondiera a un *frame*, por lo que se necesitó de una transformación en el objeto DataFrame anterior. Se empleó la librería pandas con su función `pivot` para conseguir esto. Lo anterior se puede ver en el Listado 2: en las líneas 2 a 4 se da la rotación. Tras esta operación se guardaron las coordenadas como listas dentro de cada celda, por lo que fue necesario añadir las líneas 7-9 para conseguir una tabla plana.

Python

```

1 def pivot_csv_table(video_data):
2     video_data_wide = video_data.pivot_table(
3         index="frame", columns="landmark_id", values=["x", "y", "z"]
4     )
5
6     # flatten multi-index columns
7     video_data_wide.columns = [
8         f"{coord}_{lm}{lm}" for coord, lm in video_data_wide.columns
9     ]
10    video_data_wide.reset_index(inplace=True)
11    return video_data_wide

```

Listing 2: Código para rotar la Tabla 4.4 en la que se guardan las coordenadas.

A continuación se suavizaron las coordenadas inferidas mediante un filtro de Savitzky-Golay como se indicaba en la Subsubsección 4.1.1 y definido con los mismos parámetros: un polinomio de segundo orden con una ventana de 33 elementos.

Y para terminar esta etapa, se computaron los ángulos de flexión de cada articulación. Para obtener estos ángulos se confeccionó una función que tomados tres puntos en orden podía calcular el ángulo entre las dos rectas descritas. A esta función se pasaron tres *landmarks* adyacentes capaces de definir el ángulo de flexión de la articulación deseada.

La nomenclatura empleada fue la siguiente (y como se puede ver en la Figura 4.15): a cada dedo se le asignó una letra griega, para el pulgar se definieron dos ángulos de flexión con numeración romana, siendo I la articulación interfalángica del pulgar y el II a MCP del mismo dedo. Para el resto de dedos se definió un ángulo más y estos siguieron la numeración, I para PIP, II para DIP y por último III para MCP. El cálculo del ángulo MCP de todos los dedos a excepción del pulgar fue calculado utilizando el marcador 0 de la mano como primer punto.

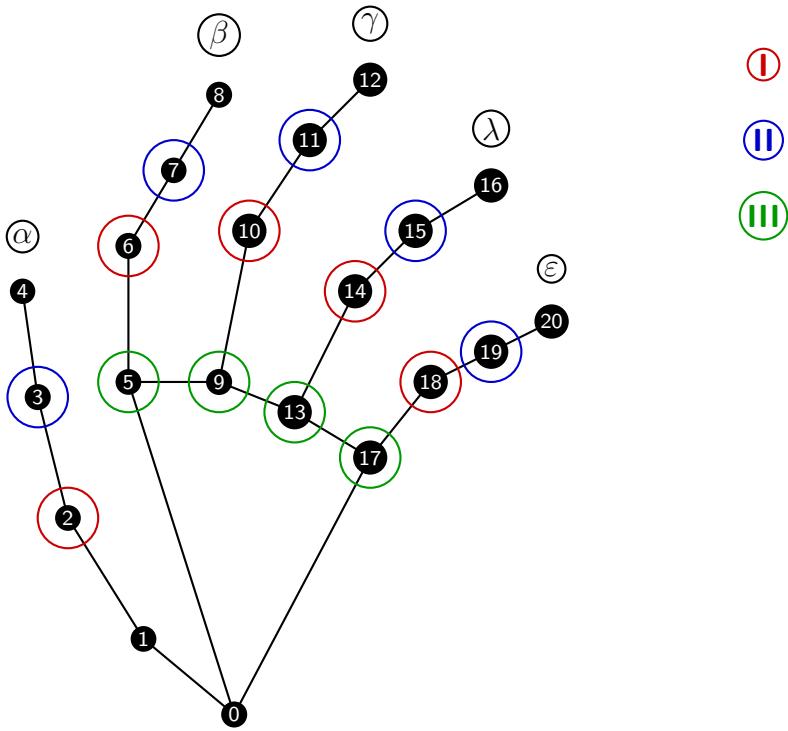


Figura 4.15: Wireframe de la mano con los *landmarks*. Se indica también la nomenclatura empleada para nombrar los ángulos extraídos.

Con esto nos queda un fichero en formato .csv con la estructura que se puede ver en la Tabla 4.4. Conseguimos una tabla en la que cada fila corresponde a un *frame* y que está listo para fusionarse más adelante con la tabla obtenida de la señal de EMG.

Tabla 4.4: Ejemplo de fichero .csv donde se almacenan los datos de las coordenadas de la mano.

frame	lm_x0	lm_x1	...	lm_z20	alfa_I	alfa_II	...	epsilon_II	epsilon_III
0	0.12	0.08	...	-0.04	0.25	-0.87	...	0.55	-1.20
1	0.14	0.09	...	-0.03	0.20	-0.80	...	0.50	-1.15
2	0.15	0.10	...	-0.02	0.18	-0.78	...	0.48	-1.10
:	:	:		:	:	:		:	:

4.2.2 Procesado de señal EMG

El procedimiento descrito a continuación se corresponde con las líneas 2 y 11 del Algoritmo 3. Este proceso fue preparado para importar los datos en forma binaria y vectorizado de BrainVision Analyzer (en un fichero .dat), por lo que se empezó importando dicho archivo con la ayuda de numpy. Todo el

proceso de filtrado de la señal realizado en BrainVision Analyzer podría replicarse en Python, pero para conveniencia del desarrollo del proyecto y puesto que no es el objetivo principal, se mantiene el filtrado realizado desde la aplicación.

La función empleada para importar la señal es la que se describe en el Listado 3. Esta función toma como entrada la dirección del fichero y el número de canales. Se establece por defecto a 5 para trabajar con los 10 canales binarizados, sin embargo, abre la ventana a importar los 10 canales por separado para aplicar los filtros pertinentes.

Dentro del Listado 3, en la línea 2 se importan los datos como un vector que almacena los canales consecutivamente y en la línea 3 se le da la forma que se puede ver en la Tabla 4.5.

Python

```

1 def emg_import(file_path, NUM_CHANNELS=5):
2     data_emg = np.fromfile(file_path, dtype=np.float32)
3     data_emg = data_emg.reshape((NUM_CHANNELS, data_emg.size // NUM_CHANNELS))
4
5     return data_emg

```

Listing 3: Código para importar la señal electromiográfica y darle la forma correcta.

A partir de la anterior estructura de datos, se trunca la señal de EMG en base a la longitud de la tabla de coordenadas del vídeo. Esta programación se recoge en el Listado 4 en el que la función definida toma como entrada los datos importados de EMG, los de video y otros dos valores para los FPS del vídeo junto con la frecuencia de muestreo de la señal, definidos por defecto a lo que se indica en el estudio. Devuelve una estructura análoga a la de entrada pero recortada.

Desde la línea 3 hasta la línea 5 del Listado 4 se comprueba que la relación entre vídeo y la señal de EMG sea la correcta. Con la condición de la línea 5 comprueba esto y además comprueba que sea una relación de números enteros. Esto es así porque al tener una relación de números enteros será posible asociar a cada *frame* un mismo número de impulsos. En caso afirmativo no es necesario ningún tipo de recorte, pero en caso negativo, se muestra por pantalla un aviso y se procede al truncamiento.

Este truncamiento se da desde la línea 14 a la 16 y simplemente convierte la duración del vídeo a segundos para poder calcular el número de muestras totales de la señal de EMG y finalmente acceder a todos los valores almacenados hasta ese máximo.

Como resultado se consiguió una estructura con las señales importadas en Python y con una longitud compatible (relacionada por un número entero) con la longitud de los datos disponibles del vídeo, obteniéndose datos parejos entre las señales de EMG y los registros procesados de los ángulos de flexión de los dedos, procedentes de los videos grabados, cuya fusión se explica en el apartado siguiente..

```
Python
```

```

1 def trimm_emg(emg_data, video_data, FPS=32, FS=1024):
2     # check if trimming required and propose automatic trimming
3     expected_ratio = FS / FPS
4     actual_ratio = emg_data.shape[1] / video_data["frame"].nunique()
5     if np.isclose(actual_ratio, expected_ratio) and actual_ratio.is_integer():
6         print("INFO: The datasets are aligned with a natural number relation.")
7         emg_data_trimmed = emg_data
8     else:
9         print(
10             """WARNING: The datasets are misaligned or have fractional relation.
11                 Manual trimming is recommended, but some will be done by deleting
12                 the overflow at end of EMG data."""
13         )
14     duration = video_data["frame"].nunique() / FPS
15     expected_emg_samples = int(duration * FS)
16     emg_data_trimmed = emg_data[:, :expected_emg_samples]
17 return emg_data_trimmed

```

Listing 4: Código para recortar el final de la señal de EMG y adaptarla a los datos disponibles del vídeo.

Tabla 4.5: Ejemplo de tabla en la que se guardan los valores de los impulsos eléctricos de la EMG.

ch_1	22.4	19.1	25.3	18.5	23.4	...
ch_2	-15.6	-12.3	-11.4	-18.0	-16.7	...
ch_3	18.7	22.9	15.6	20.4	17.9	...
ch_4	31.2	29.4	33.1	27.9	30.5	...
ch_5	-12.8	-10.5	-14.7	-13.0	-12.7	...

4.2.3 Fusión de datos

Para fusionar los datos en un mismo DataFrame se utilizó el comando `pandas.concat` tal y como se ven en el Listado 5.

```
Python
```

```

1 X = pd.concat(
2     [
3         video_data.reset_index(drop=True),
4         emg_data,
5     ],
6     axis=1,
7 )

```

Listing 5: Comando para concatenar ambos DataFrame.

Todas las líneas de código comentadas en esta sección se encuentran en el Apéndice D y también en el repositorio de Github del proyecto.

Con todo esto, se exportó con ayuda de pandas el DataFrame a .csv para entrenar el algoritmo

Tabla 4.6: Estructura final dentro del .csv de la base de datos.

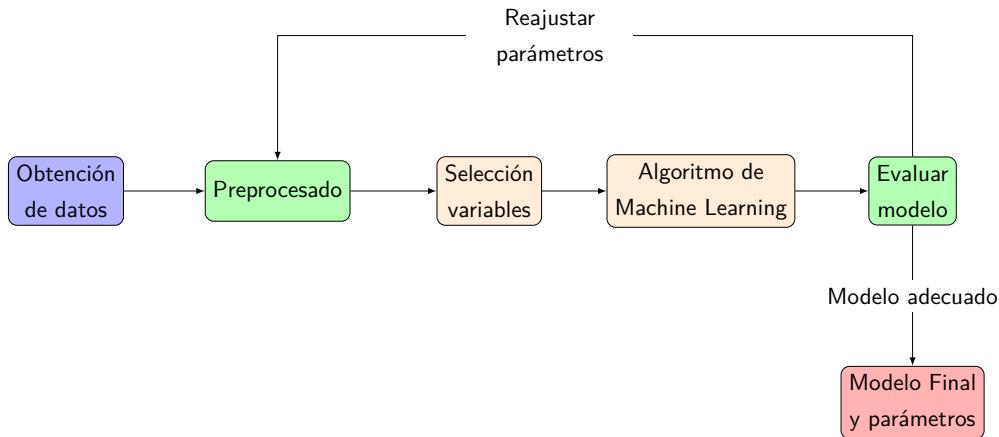
frame	lm_x0	lm_x1	...	epsilon_ll	epsilon_III	emg_0_ch_1	...	emg_30_ch_5	emg_31_ch_5
0	0.12	-0.07	...	0.56	-1.21	0.02	...	-0.03	0.01
1	0.11	-0.06	...	0.54	-1.18	0.05	...	-0.04	0.00
2	0.13	-0.08	...	0.57	-1.16	0.01	...	-0.02	0.02
:	:	:	⋮	⋮	⋮	⋮	⋮	⋮	⋮

de ML en la próxima sección.

4.3 Obtención del modelo

Para el entrenamiento se trabajó en Google Colab debido a la posibilidad de usar sus servidores para entrenar el algoritmo. Usar computadores remotos es una práctica habitual en ámbitos como el del entrenamiento de algoritmos de ML o la simulación por métodos numéricos de sistemas físicos.

Por lo general, en un proceso de entrenamiento de modelos por ML se sigue un *pipeline* u hoja de ruta que se representa en la Figura 4.16 y consiste en los siguientes pasos: preprocessado de los datos, selección de propiedades, entrenamiento del modelo y evaluación del modelo. El proceso se itera hasta que se logra un modelo apropiado con el error deseado.

**Figura 4.16:** Representación del *pipeline* seguido habitualmente en procesos de aprendizaje automático.

En este proyecto se ha incluido parte del preprocessado dentro del apartado anterior (al suavizar las coordenadas y extraer los ángulos de flexión), por lo que sólo queda la normalización del movimiento de los ángulos para facilitar la tarea de los algoritmos de predicción. Con esto se pasó al apartado de selección de variables.

4.3.1 Selección de variables

Para obtener el mejor modelo predictivo se buscó el conjunto de variables más apropiado para el entrenamiento del mismo. Las posibilidades estudiadas fueron las que se pueden ver en la Tabla 4.7, en la que se alternó si se introduce la señal de EMG junto con los ángulos del *frame* anterior (θ_{-1}) o no. También se valoró predecir la diferencia de ángulos con el *frame* anterior ($\Delta\theta$) a partir de la señal de EMG. Por lo que en el estudio a continuación se repitió tres veces el proceso de generar modelo para contrastar las estrategias y sus resultados.

Tabla 4.7: Posibles combinaciones de los datos para entrenar al modelo.

Datos (entrada)	Etiquetas (salida)
EMG	θ
EMG + θ_{-1}	θ
EMG	$\Delta\theta$

4.3.2 Entrenamiento del modelo

Para entrenar el modelo se probó en un principio a hacer una CNN y también una ANN como estudiaron Shirao, Reddy y Kosuri [8]. Para ello se empleó la librería tensorflow y su API para Python. También se añadió un modelo transformador con atención con la intención de evaluar si la atención es un factor importante.

La estrategia seguida para la ANN fue iniciar con una capa densa del tamaño de la entrada, luego dos capas densas con una cantidad de neuronas $3/2$ mayor que la entrada todas con una función de activación con tipo ReLU. La red terminó con otra capa densa con tantas neuronas como ángulos y una activación lineal. Este modelo se puede ver en el siguiente Listado 6.

La estrategia para la CNN fue similar a la anterior, comenzando con una capa de entrada del tamaño adecuado para los datos. A continuación se añadió una capa convolucional con `in_size` filtros y activación ReLU. Posteriormente se incluyó una capa de *max pooling* para reducir la dimensionalidad y aumentar la robustez frente al *overfitting*. La arquitectura continuó con una segunda capa convolucional con $3/2$ veces más filtros que la anterior, seguida nuevamente de una capa de *pooling*. Los datos resultantes se aplanan antes de una capa densa con el mismo número de unidades que la última convolución. Para prevenir el sobreajuste, se aplica un Dropout del 50%, y finalmente se añadió una capa densa de salida con tantas neuronas como ángulos se desea predecir.

Con lo anterior, se entrenaron seis modelos que se compararon según su desempeño de los diferentes

modelos según el tipo de *dataset* proporcionado.

Finalmente, se probó un último modelo transformador con atención y capaz de entender cada uno de los canales de EMG como uno en lugar de los 32 impulsos eléctricos asignados a cada *frame*. La construcción de este modelo fue la que se puede ver en el Listado 8. La creación del modelo se siguió de esta forma para conseguir una descripción paramétrica del mismo, ofreciendo la posibilidad de cambiar valores como el número de capas o ajustar la entrada y salida según se necesite al momento de llamar al objeto. Se empleó una nomenclatura de clases, a diferencia de la anterior, en la que se definieron cada una de las capas y parámetros como métodos dentro de la clase.

Python

```

1 in_size = X_train.shape[1]
2
3 model_ann_X = Sequential([
4     Input(shape=(in_size, 1)),
5     Dense(in_size, activation='relu', input_shape=(X_train.shape[1], 1)),
6     Dense(int(3*in_size/2), activation='relu'),
7     Dense(int(3*in_size/2), activation='relu'),
8     Dense(y_train.shape[1], activation='linear')
9 ])

```

Listing 6: Definición del modelo de ANN y sus capas con tensorflow.

Python

```

1 in_size = X_train.shape[1]
2
3 model_cnn_x = Sequential([
4     Input(shape=(in_size,1)),
5     Conv1D(in_size, kernel_size=3, activation='relu'),
6     MaxPooling1D(pool_size=2),
7     Conv1D(int(3*in_size/2), kernel_size=3, activation='relu'),
8     MaxPooling1D(pool_size=2),
9     Flatten(),
10    Dense(int(3*in_size/2), activation='relu'),
11    Dropout(0.5),
12    Dense(y_train.shape[1], activation='linear')
13 ])

```

Listing 7: Definición del modelo de CNN y sus capas con tensorflow.

```
Python
```

```

1 class EMGTransformer(tf.keras.Model):
2     def __init__(self,
3                  seq_len=32,
4                  input_dim=5,
5                  output_dim=15,
6                  d_model=64,
7                  n_heads=4,
8                  ff_dim=128,
9                  num_layers=2,
10                 ):
11         super(EMGTransformer, self).__init__()
12         self.seq_len = seq_len
13         self.input_proj = layers.Dense(d_model)
14
15         self.pos_encoding = positional_encoding(seq_len, d_model)
16         self.encoder_layers = [
17             layers.MultiHeadAttention(num_heads=n_heads, key_dim=d_model)
18             for _ in range(num_layers)
19         ]
20
21         self.ff_layers = [
22             models.Sequential(
23                 [layers.Dense(ff_dim, activation="relu"), layers.Dense(d_model)])
24             )
25             for _ in range(num_layers)
26         ]
27         self.norm_layers = [
28             layers.LayerNormalization(epsilon=1e-6) for _ in range(num_layers)
29         ]
30
31         self.global_pool = layers.GlobalAveragePooling1D()
32         self.output_layer = layers.Dense(output_dim)
33         return self.output_layer(x)

```

Listing 8: Definición del modelo con atención y sus capas con tensorflow.

4.3.3 Evaluación de los modelos

Para evaluar los modelos se empleó el MAE ya que al trabajar en radianes, el error vendrá dado por números bajos (menores de 6,28). Se incorporaron también los valores de RMSE para completar la valoración del modelo, ya que éste último mostró información acerca de si el error se encuentra equilibrado y/o distribuido homogéneamente, lo que es fundamental para estimar la viabilidad del modelo. En la librería de tensorflow, esto se define al definir el compilador del modelo (que será el mismo para todos los modelos neuronales estudiados) que se puede ver en Listado 9. El optimizador tipo adam es el más usado y la opción por defecto, por lo que se optó por mantener esta opción.

Además se representó la evolución de los ángulos a lo largo del tiempo junto con su predicción. En el caso de aquellos modelos que empleaban información del ángulo anterior, se proporcionó un punto de partida y a partir de ahí se realimentó al modelo de predicción con sus propias predicciones. Esto mostró de una forma más real el desempeño del modelo y permitió comprobar si algún modelo poseía una salida aleatoria pero capaz de reducir el error.

Python

```

1 model.compile(optimizer='adam',
2                 loss='mae',
3                 metrics=[tf.keras.metrics.RootMeanSquaredError()])

```

Listing 9: Definición del compilador para los modelos de redes neuronales de tensorflow.

4.3.4 Estudio de la evolución del modelo

Este estudio se enfocó como piloto para la técnica desarrollada, por lo que la N (tamaño muestral) seleccionada fue arbitraria y con el objetivo de determinar el tamaño real necesario. Por eso fue interesante estudiar como afectó la adición de nuevos vídeos y nuevos movimientos en la mano al entrenamiento del modelo y su error. Para ello, se entrenó en serie el mismo modelo (*transformer* con atención) añadiendo un nuevo vídeo cada iteración. En cada iteración se evaluó el modelo y obtuvo un valor del MAE y RMSE que se recogió y se representó en una gráfica, mostrando la evolución de los errores conforme se añadieron más vídeos.

4.4 Conectividad con Arduino

Uno de los objetivos finales del PFG fue el de controlar una prótesis, este proceso se planteó sobre un chip capaz de realizar los cálculos matemáticos necesarios para aplicar el modelo de ML generado y predecir nuevas posiciones a partir de la señal del paciente. En esta sección se plantea un código que permite a un controlador basado en programación en C++ como puede ser Arduino recibir y leer los datos a través de puerto serie de otro procesador que genera las predicciones.

Para ello se necesitó un primer *script* en Python que predijo en base a la EMG obtenida y envió los datos por puerto serie al Arduino. Luego, desde el controlador se recibieron los datos y se interpretaron para enviar a cada actuador. Es necesario, si se quiere implementar, que se calcule la transformación necesaria para ajustar los ángulos obtenidos a rotación (o desplazamiento) de los actuadores.

Para cargar el modelo se necesitó definir la estructura empleada para el modelo, esto se puede ver en

las líneas 6-8 del Listado 10. Se omitió la definición del modelo que se puede ver en el Listado 8. Tras esto se cargó el modelo (alojado en la misma carpeta que el *script*) y se comenzó la comunicación por puerto serie. Es necesario ajustar la dirección (en el ejemplo, COM3) del Arduino según el puerto usado (véase líneas 10 y 11 del Listado 10).

Finalmente, mediante un bucle se obtienen y se preparan las señales EMG para introducir al modelo. Después, las predicciones se envían a través de un hilo de texto codificado para que después el controlador pueda recibir correctamente y decodificarlo. Las funciones de las líneas 14 y 15 deberán ser programadas por el usuario para recibir la señal del sensor de EMG y para tratar y preparar la señal para que tenga la forma apropiada para el modelo.

Python

```
1 import tensorflow as tf
2 from tensorflow.keras import layers, models, saving
3 import numpy as np
4 import serial
5
6 @tf.keras.utils.register_keras_serializable()
7 class EMGTransformer(tf.keras.Model):
8     # ...
9
10 model = load_model("./modelo.keras")
11 arduino = serial.Serial('COM3', 9600)
12
13 while True:
14     emg_signal = obtener_emg()
15     emg_signal = filtrar_emg(emg_signal)
16     angles = model.predict(emg_signal)[0]
17
18     data_string = ",".join([f"{angle:.2f}" for angle in angles])
19     arduino.write((data_string + "\n").encode())
20
21 def obtener_emg():
22     # ...
23
24     return emg_signal
25
26 def filtrar_emg(emg_signal):
27     # ...
28
29     return emg_signal_filtrada
```

Listing 10: Código para predecir los ángulos de las articulaciones de la mano partir de EMG y enviar a un controlador por puerto serie.

Tras esto, se muestra el código mínimo en C++ para recibir las órdenes. Para implementarlo en un controlador Arduino será necesario rellenar el código con las funciones pertinentes en los lugares que en este código se representa mediante puntos suspensivos en comentarios.

Se comenzó inicializando la variable que almacena las órdenes recibidas en la línea 4 del Listado 11. Luego se inició la comunicación por puerto serie a 9600 baudios (línea 10). El bucle principal es similar a lo que se describió en Algoritmo 2, en este caso se representa en código C++ en lugar de pseudocódigo. Se comprueba si el puerto serie está disponible y se lee la información, si no se ha encontrado el final del hilo (denotado por \n) se concatena el valor. La operación += adquiere la función de concatenar al final de una variable String en lenguajes como C++. Una vez encontrado el carácter final se hace llamada a la función que se encargada de desencriptar la orden. (Véase líneas 14-22 del Listado 11).

Dentro de la función para procesar la orden, se busca el número (en formato String) entre dos delimitadores, que en este caso fueron comas. Luego se convierte en un número en formato de coma flotante (float) y se almacena en el vector de los ángulos. Una vez se acaba de procesar toda la línea de texto introducida desde el bucle principal se llama a la función para actualizar el estado de los actuadores. Esta función es la que se debe de customizar para el equipo utilizado.

Además, se recomienda dejar el bucle principal intacto, ya que al introducir comandos se podría obstaculizar la función de lectura de órdenes del puerto serie, lo que podría derivar en el salto de órdenes.

C++

```
1 /*  
2 ...  
3 */  
4 String input = "";  
5  
6 void setup() {  
7     /*  
8     ...  
9     */  
10    Serial.begin(9600);  
11 }  
12  
13 void loop() {  
14     if (Serial.available()) {  
15         char c = Serial.read();  
16         if (c == '\n') {  
17             processInput(input);  
18             input = "";  
19         } else {  
20             input += c;  
21         }  
22     }  
23     /*  
24     ...  
25     */  
26 }  
27  
28 void processInput(String input) {  
29     float angles[5];  
30     int idx = 0;  
31     int lastComma = -1;  
32  
33     for (int i = 0; i < input.length(); i++) {  
34         if (input.charAt(i) == ',' && i == input.length() - 1) {  
35             String value = input.substring(lastComma + 1, (i == input.length() - 1) ? i + 1 : i);  
36             angles[idx++] = value.toFloat();  
37             lastComma = i;  
38         }  
39     }  
40     executeInput(angles);  
41 }  
42  
43 void executeInput(float angles) {  
44     /*  
45     ...  
46     */  
47 }
```

Listing 11: Código para la lectura de las órdenes provenientes del modelo.

5 Resultados y discusión

5.1 Evaluación de la base de datos

En esta sección se presenta la base de datos obtenida como uno de los resultados del proyecto. Se busca explicar las características de esta base de datos y su estructura. No se hablará de la estructura del fichero .csv como se ha hecho previamente en la Tabla 4.6 de la Sección 4.2.

La longitud de los vídeos obtenidos junto con los movimientos-tipo presentes se presentan en la siguiente Tabla 5.1. Los movimientos-tipo recogidos en dicha tabla fueron los siguientes:

1. Apertura/cierre de la mano sencillo y sucesivo.
2. Movimiento de los dedos de forma individual.
3. Apertura o cierre de la mano mantenido en el tiempo, inmóvil.

Estos tipos de movimiento se eligieron con la intención de proporcionar al modelo formas de abstraer la flexión/extensión de los dedos de forma independiente. Además se introdujo la presencia de movimientos mantenidos, ya que de no ser así, el modelo predictivo entendería que una mano debe estar en movimiento constante.

En total se obtuvieron 100.255 posturas de la mano que se relacionaron con 32 registros EMG de 5 canales. Resultando en un sistema que tomaba 160 parámetros de entrada para predecir 14 de salida. Por lo general, al entrenar un modelo siempre se busca que la cantidad de datos disponibles supere en gran medida a la cantidad de parámetros de entrada. Dada la cantidad muestral en comparación de los datos de entrada se estimó que era suficiente para llevar a cabo este proyecto como un estudio preliminar fiable.

Tabla 5.1: Características de los vídeos adquiridos durante la confección de la base de datos.

Vídeo	Duración [min:seg] ([frames])	Movimientos
1	3:01.93 (5822)	1
2	2:17.84 (4411)	1
3	1:00.40 (1933)	1
4	2:46.87 (5340)	1
5	4:57.31 (9514)	1
6	6:00.31 (11530)	1 + 2
7	4:37.53 (8881)	1 + 2
8	3:33.71 (6839)	1 + 2
9	4:40.93 (8990)	1 + 2
10	4:47.59 (9203)	1 + 2 + 3
11	6:46.90 (13021)	1 + 2 + 3
12	7:41.59 (14771)	1 + 2 + 3
Total	52:12.96 (100255)	

5.2 Evaluación de los modelos

De los modelos generados en el capítulo anterior, en la Subsección 4.3.2, se obtuvieron los siguientes indicadores que se reflejan y se comparan en la Tabla 5.2, en la que se presentan los errores al momento del entrenamiento, durante la validación. En la Tabla 5.3 se muestran los mismos indicadores de error obtenidos al realizar un test con una serie de datos nunca vistos por el modelo. Junto al valor obtenido en radianes se da el valor en grados sexagesimales para una interpretación más visual. Las distintas entradas y salidas que se presentan en los modelos fueron las que se explican en la Subsección 4.3.1.

La intención de recoger el valor de MAE y de RMSE fue la de analizar la relación entre los errores bajos y los más altos como se explica en la Subsección 4.3.3.

De ambas Tablas 5.2 y 5.3 se puede extraer que los modelos predictivos con menor error fueron los que recibían información de la posición anterior. Mientras que el modelo que más error obtuvo fue el *transformer*.

Durante el estudio también se pudo ver que el error no fue el mismo para todas las articulaciones. A continuación se muestra en las siguientes Figuras 5.1 y 5.2 los errores separados para cada articulación en relación con su rango de flexión. Es decir, en el eje *x* se indica la articulación como se indicaba en la Figura 4.15 (donde se explica la nomenclatura empleada en la base de datos para nombrar los ángulos de flexión) y en el eje *y* se indica en porcentaje el MAE obtenido para cada articulación.

Además, en las mismas gráficas se representan con una línea punteada el valor del MAE general de

cada modelo ajustado al rango de flexión. Esto se hace así con la intención de comparar el error del modelo predictivo con el error individual de cada articulación.

Los rangos de flexión empleados para el calculo del porcentaje fueron los que recogen Schuenke [25] y Hochschild [26] en sus manuales:

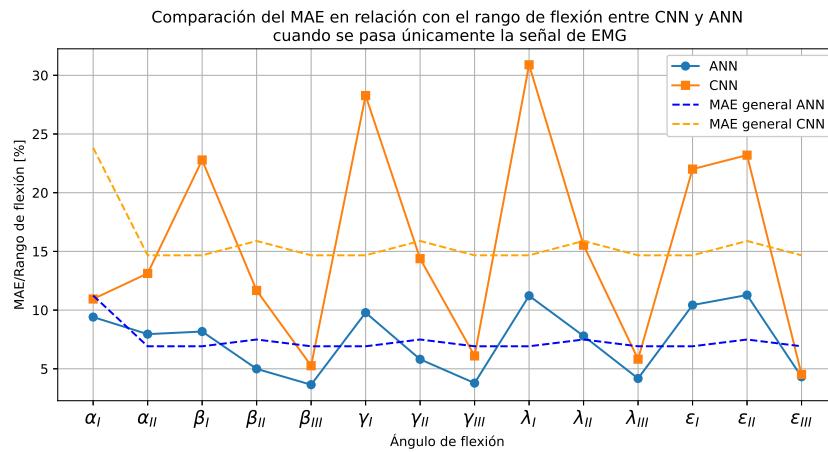
- MCP del pulgar (α_I): 80°de flexión y 0°de extensión. En total se usó un rango de 80°.
- Resto de MCP ($\beta_{III}, \gamma_{III}, \lambda_{III}, \varepsilon_{III}$): 90°de flexión y 40°de extensión. En total se usó un rango de 130°.
- PIP ($\alpha_I, \beta_I, \gamma_I, \lambda_I, \varepsilon_I$): 130°de flexión y 0°de extensión. En total se usó un rango de 130°.
- DIP ($\beta_{II}, \gamma_{II}, \lambda_{II}, \varepsilon_{II}$): 90°de flexión y 30°de extensión. En total se usó un rango de 120°.

Tabla 5.2: Valores obtenidos de MAE y RMSE al final del entrenamiento del modelo (validación).

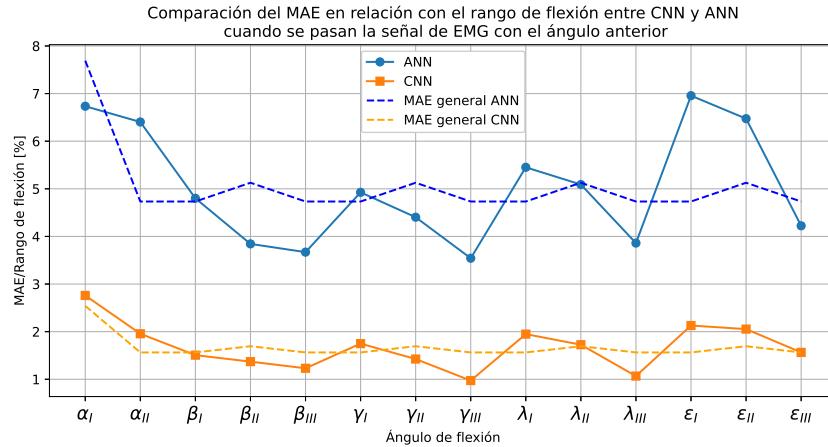
Modelo	Entrada	Salida	MAE [rad] ([°])	RMSE [rad] ([°])
CNN	EMG	θ	0.326 (18.678°)	0.4704 (26.952°)
	EMG + θ_{-1}	θ	0.0360 (2.0626°)	0.0566 (3.2429°)
	EMG	$\Delta\theta$	0.1409 (8.073°)	0.2788 (15.974°)
ANN	EMG	θ	0.158 (9.0527°)	0.258 (14.782°)
	EMG + θ_{-1}	θ	0.1091 (6.251°)	0.1635 (9.3679°)
	EMG	$\Delta\theta$	0.4492 (25.737°)	0.6773 (38.806°)
Transformer	EMG	θ	0.8734 (50.0421°)	1.1558 (66.2225°)

Tabla 5.3: Valores obtenidos de MAE y RMSE al probar (test) los modelos entrenados.

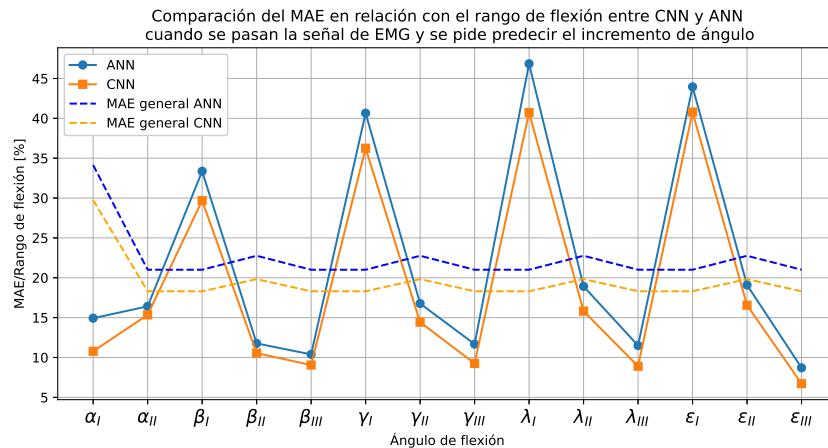
Modelo	Entrada	Salida	MAE [rad] ([°])	RMSE [rad] ([°])
CNN	EMG	θ	0.3327 (19.062°)	0.4792 (27.456°)
	EMG + θ_{-1}	θ	0.0355 (2.034°)	0.0561 (3.2143°)
	EMG	$\Delta\theta$	0.1393 (7.9813°)	0.2758 (15.802°)
ANN	EMG	θ	0.1569 (8.9897°)	0.257 (14.725°)
	EMG + θ_{-1}	θ	0.1074 (6.1536°)	0.1615 (9.2533°)
	EMG	$\Delta\theta$	0.4409 (25.262°)	0.6699 (38.382°)
Transformer	EMG	θ	1.0208 (58.4875°)	1.2219 (70.0097°)



(a) Comparación entre CNN y ANN cuando solo se pasan datos de la EMG.



(b) Comparación entre CNN y ANN cuando se pasa la EMG junto al valor del ángulo del frame anterior.



(c) Comparación entre CNN y ANN cuando solo se pasan datos de la EMG y se pide que prediga el incremento del ángulo con respecto al frame anterior.

Figura 5.1: Comparación del MAE en proporción con el rango de flexión de la articulación entre los dos modelos para cada tipo de dataset.

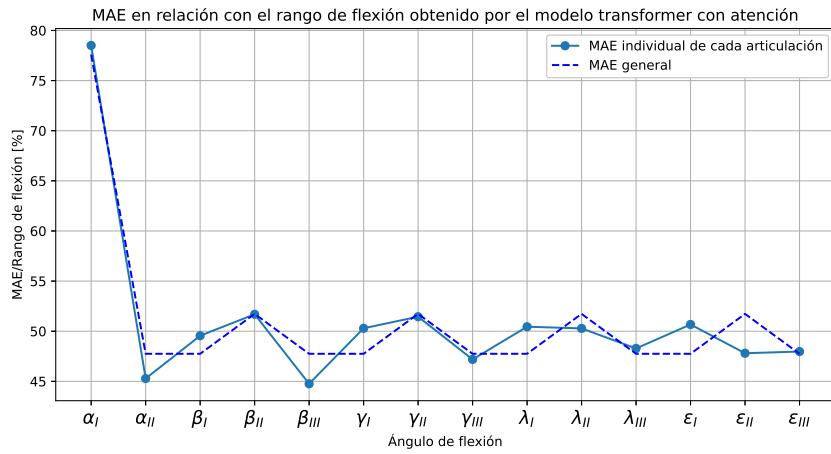


Figura 5.2: MAE obtenido en proporción con el rango de flexión de las articulaciones de la mano

En la Figura 5.1 se puede ver que de los tipos de set de datos proporcionados, el que produce los peores modelos es aquel que devuelve como salida el incremento del ángulo (véase Figura 5.1c), como también se veía en las Tablas 5.2 y 5.3, en las que se recogen los errores de validación y test.

En cuanto a la comparación de los modelos sin atención y con atención, y de acuerdo con la Figura 5.1a, hay una mejor predicción en el modelo ANN cuando el set de datos o variable introducida, es directamente los registros de EMG. Sin embargo, si el set de datos proporcionado al modelo es el basado en el ángulo anterior, el mejor modelo de predicción pasa a ser el CNN, como recoge la Figura 5.1b.

Todos los modelos sin atención presentan mayor dificultad a la hora de seguir los movimientos de las articulaciones afectadas por los flexores superficiales (DIP o marcados con subíndice I en la Figura 4.15) a excepción del pulgar (α). De la misma forma, todos los modelos sin atención, presentaron valores de MAE mas bajos en la predicción de los ángulos de flexión de la zona MCP (articulaciones con los subíndices III).

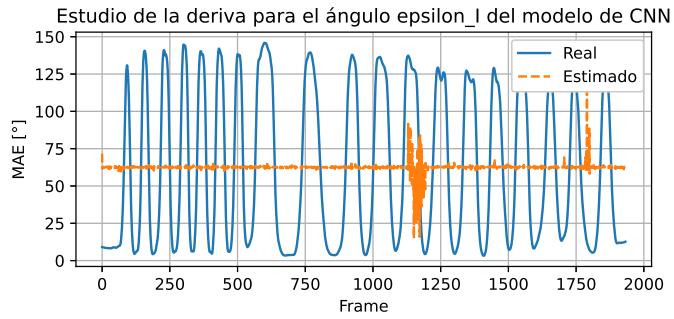
En el caso del modelo *transformer* con atención (Figura 5.2), el ángulo mejor predicho de la mayoría de dedos es el descrito por la articulación PIP, a diferencia de los que no tienen atención. En el caso del pulgar se dispara el error en el ángulo de flexión de la articulación MCP. Este valor es debido a que muchos de los músculos involucrados en la movilidad del pulgar se encuentran dentro de la mano y no llegan hasta el punto de la toma de datos en el antebrazo. La medio de los errores obtenidos con respecto al rango de flexión expresado en porcentaje es de 51 %

En resumen, se puede extraer que aquellos ángulos de flexión que involucran músculos más fuertes y más cercanos a la piel, proporcionan a los modelos de predicción unas lecturas de EMG más limpias produciendo predicciones con menos error.

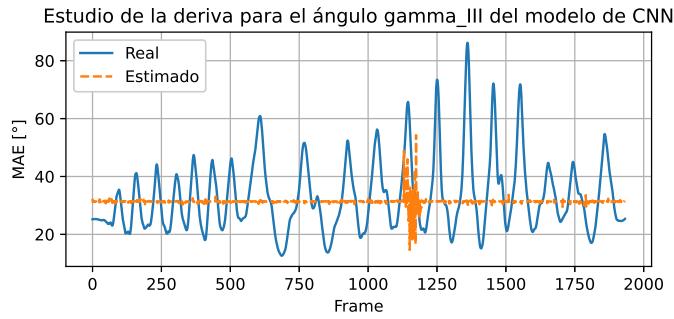
Finalmente, se presenta en las Figuras 5.3 y 5.4 las predicciones obtenidas en comparación con su valor real. Se elige representar los ángulos de flexión que presentan el valor de MAE más alto, y el más bajo, con objeto de evaluar la calidad del modelo en un caso aplicado y visualizar de forma gráfica el origen de los valores de MAE y RMSE obtenidos.

Además, este procedimiento sirve en el caso del modelo de CNN con información del ángulo previo para comprobar si aparece un error acumulativo debido a imprecisiones, también conocido como derivado o *drift*.

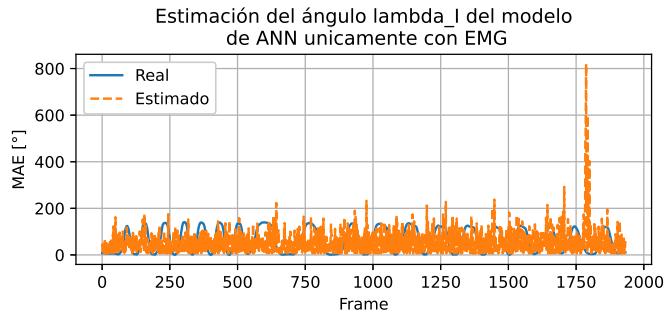
De esta forma, el eje *x* de las Figuras 5.3 y 5.4 representa el tiempo en *frames* de vídeo y el eje *y* representa el valor del ángulo.



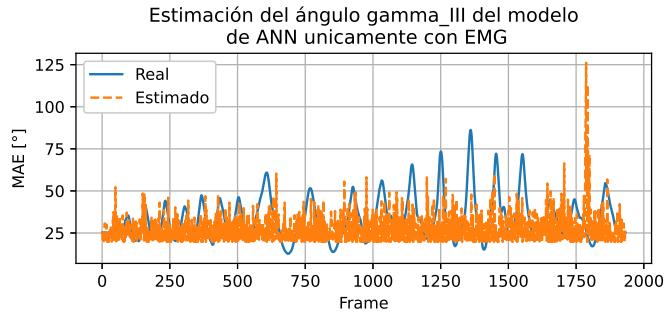
(a) Estudio de la deriva para el ángulo de mayor error, ϵ_I del modelo de CNN.



(b) Estudio de la deriva para el ángulo de menor error, γ_{III} del modelo de CNN.



(c) Estimación del ángulo con mayor error, λ_I , del modelo de ANN únicamente con EMG.



(d) Estimación del ángulo con menor error, γ_{III} , del modelo de ANN únicamente con EMG.

Figura 5.3: Estudio de la presencia de deriva en las predicciones y contraste de la precisión para la predicción entre CNN cuando se da información sobre la posición anterior y ANN cuando solo recibe la señal de EMG.

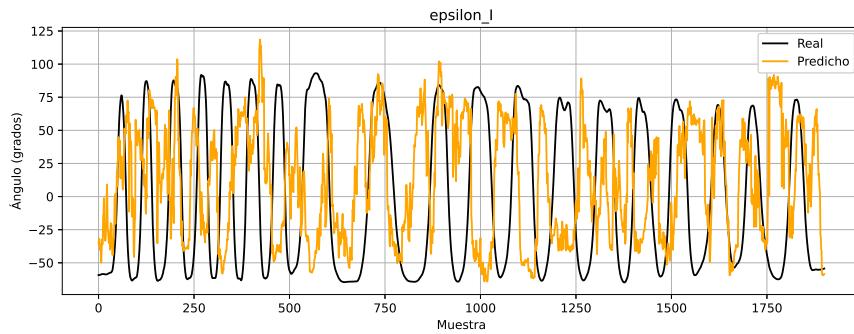
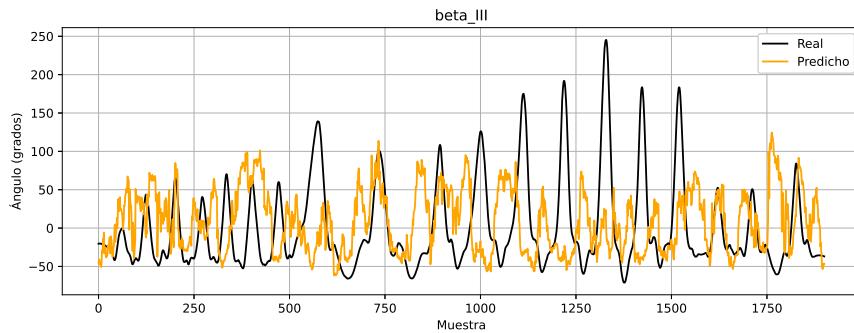
(a) Estimación del ángulo con mayor error, ε_I , del modelo transformador con atención.(b) Estimación del ángulo con menor error, ε_I , del modelo transformador con atención.

Figura 5.4: Estudio de la precisión de la predicción del modelo transformador con atención cuando se le pasa la señal de EMG comparado con el movimiento real del dedo para datos nunca vistos.

Se puede ver en la Figura 5.3 que los modelos sin atención no son capaces de abstraer el movimiento de la mano y el motivo de su bajo error viene de una optimización basada en situar puntos inconexos que reducen el valor del error. Pero al representar gráficamente, se ve claramente que las predicciones no describen un movimiento factible para el control de una prótesis.

En cambio, en el caso del modelo *transformer* con atención, se ve que este consiguió abstraer la naturaleza del movimiento de la mano humana, dibujando líneas continuas y a menudo suaves cercanas a un movimiento natural. Sin embargo, el valor alto del error recogido en las Tablas 5.2 y 5.3, se explica al ver como no consigue que las curvas predichas se solapen con las reales.

5.3 Evaluación de escalabilidad del estudio

El tamaño del set de datos disponible es algo que condiciona la calidad de los modelos entrenados en cualquier disciplina que emplee algoritmos de ML. Es por eso que se estudió la evolución del error del modelo a medida que se le iban pasando más vídeos y se iba aumentando el tamaño del set de datos. Esta prueba se llevó a cabo para el modelo *transformer* con atención. El orden de los vídeos introducidos es el que se sigue en la Tabla 5.1, en la que se muestran las dimensiones de la base de datos. Se emplearon todos los vídeos en el estudio a excepción del segundo vídeo que fue el que se usó para el test con datos nunca vistos. A continuación en la Figura 5.5 se puede ver la evolución del error.

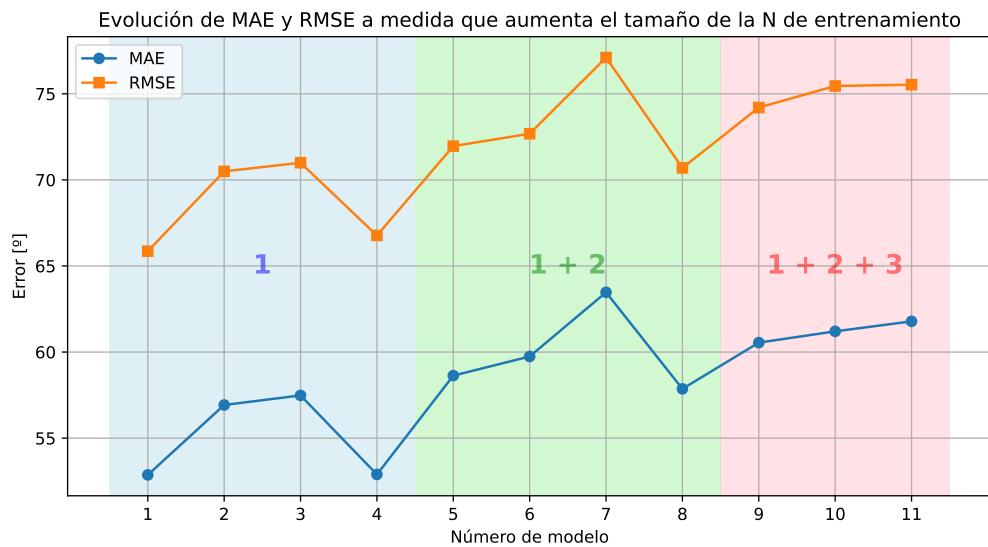


Figura 5.5: Evolución de los errores a medida que se entrena el modelo *transformer* con atención con más vídeos. Los vídeos (identificado como número de modelo) siguen el orden declarado en la Tabla 5.1, y con franjas de color se expresan los movimientos-tipo presentes en dichos vídeos.

En la Figura 5.5 se ve como el error aumenta de forma general a medida que aumenta el tamaño muestral. Se puede observar la presencia de valores más bajos inmediatamente seguidos por subidas en el error, estos valores bajos se explican cuando el modelo es capaz de generalizar los movimientos presentes en esa parte de la base de datos. El incremento repentino sucede al introducir un nuevo movimiento-tipo en la base de datos. Es decir, hasta la cuarta iteración, el modelo solo dispone del movimiento-tipo de apertura y cierre de la mano, sin embargo, en la quinta iteración se introduce el movimiento-tipo de flexión de los dedos de forma independiente, hasta la octava iteración. De nuevo, en la novena iteración se introduce un nuevo movimiento-tipo (inmovilidad de apertura y cierre de la mano) que empeora la calidad del modelo hasta que este es capaz de generalizar.

En estas iteraciones finales, se ve como el RMSE se mantiene estable mientras que el MAE crece

levemente de forma estable. Esto indica que nuestro modelo siguió mejorando, reduciendo los errores más altos, y predeciblemente lograría una reducción del error general con un mayor tamaño muestral.

6 Conclusiones

De los resultados obtenidos del proyecto, se pueden extraer las siguientes conclusiones que responden a los objetivos establecidos en la introducción.

6.1 Conclusiones fundamentales

Durante las primeras etapas del PFG se ha encontrado una forma eficaz de grabar en paralelo y combinar las señales de EMG junto con el modelo cinemático extraído de las imágenes de vídeo. Se ha concluido de forma positiva documentando un procedimiento, haciendo hincapié en aquellos dispositivos o elementos que formaron parte del sistema de adquisición, susceptibles de cambio, y que determinaron el procedimiento a seguir.

Se logró una base de datos capacitada para el entrenamiento de modelos mediante ML. Esta base de datos se compuso 52 minutos de grabaciones que se combinaron con las señales de EMG junto con los ángulos de flexión de las articulaciones del modelo cinemático de una mano. Todo organizado de tal forma que cada fila se corresponde con un instante del vídeo con su número de impulsos eléctricos asociados. Esta base de datos sirvió como piloto y demostró la viabilidad de entrenar un modelo de esta forma.

En cuanto a la automatización, se implementaron *scripts* y funciones preparadas a partir de los principios de extensibilidad y reproducibilidad. Esta automatización englobó las labores del proyecto desde la grabación en paralelo hasta la obtención de la base de datos.

Se entrenaron diversos modelos basados en redes neuronales que aprovecharon la capacidad de estos algoritmos para encontrar relaciones implícitas entre las señales de entrada y las de salida. Se concluyó que el mejor modelo fue el *transformer* con atención. Este modelo obtuvo un MAE de 50° en validación durante el entrenamiento y de 58° al probar en test. Estos valores son más altos que los obtenidos en los modelos sin atención (ANN con EMG: 9°; CNN con EMG y ángulo previo: 2°), sin embargo, al

representar los resultados de las predicciones frente al valor real del ángulo en el set de datos usado para el test, se pudo ver que el modelo con atención fue el único capaz de abstraer el movimiento suave y continuo de una mano.

Es decir, al representar gráficamente los resultados de las predicciones sobre un test de los modelos ANN y CNN, aún teniendo un MAE más bajo que el modelo *transformer* con atención, su salida adquirió la forma de una nube de puntos para minimizar el error, sin reproducir la dinámica del movimiento de la mano.

Se concluyó que deben procesarse al menos 100.255 *frames* de datos para obtener un prototipo de predicción viable. El aumento del tamaño muestral mejorará la viabilidad del modelo.

Se preparó una integración que permitió la comunicación del modelo con un microcontrolador basado en C desde Python y a través del puerto serie.

En definitiva, y como conclusión global del PFG, se ha obtenido un modelo de *transformer* con atención para predicción, capaz de obtener la posición de los dedos de una mano basándose en señales de EMG tomadas en el antebrazo. Este modelo, logra ser un modelo de regresión, permitiendo una combinación infinita de movimientos en la prótesis controlada.

6.2 Limitaciones técnicas y futuras líneas de estudio

Durante el PFG se han encontrado las siguientes dificultades, que han influido en la calidad de los resultados o en la necesidad de adaptaciones de los procedimientos.

El uso de dos ordenadores independientes para la recolección de señales en paralelo, requirió de tratamientos de los datos adquiridos para adecuar su longitud y sincronizarlos. Se estima más recomendable el uso de un sólo equipo, de forma que desde una misma aplicación (en el caso del estudio, Python) se recojan en paralelo ambas señales.

Como se comprobó a la hora de obtener la señal de EMG, el diseño de un dispositivo que sitúe los electrodos de forma normalizada y permita aplicar cierta presión aumentando el contacto, como un brazal, permitirá una mejora de la calidad de la señal, y por tanto, del modelo generado, ayudando a estandarizar la ubicación para la lectura de señal.

Durante la captura del modelo cinemático se empleó visión artificial por su accesibilidad. Sin embargo, el uso de un sistema de captura de movimiento más rígido (como triangulación con varias cámaras) podría ofrecer resultados más consistentes y precisos, abriendo la posibilidad a un estudio dinámico de

la fuerza ejercida.

Para la obtención de la base de datos se empleó únicamente un individuo, por lo que un modelo ideal requeriría del uso de muchos más individuos para poder generalizar los movimientos de la mano en toda la población.

El modelo final desarrollado en el estudio ofrece la posibilidad de aprendizaje por refuerzo, abriendo las puertas al estudio de la viabilidad de un modelo pre-entrenado que siga evolucionando y adaptándose al caso particular y movimientos un único individuo.

El principio fundamental de este estudio es perfectamente extrapolable a cualquier extremidad, siempre que se proporcione una zona para la obtención del modelo cinemático y otra para los músculos involucrados en los movimientos, para la extracción de las señales EMG.

Bibliografía

- [1] K. Cabegin, M. Lim, D. T. Fernan, R. Garcia Santos y G. Magwili, «Electromyography-based Control of Prosthetic Arm for Transradial Amputees using Principal Component Analysis and Support Vector Machine Algorithms,» en *2019 IEEE 11th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM)*, 2019, págs. 1-6. DOI: [10.1109/HNICE48295.2019.9073353](https://doi.org/10.1109/HNICE48295.2019.9073353).
- [2] V. Gohel y N. Mehendale, «Review on electromyography signal acquisition and processing,» *Biophysical Reviews*, vol. 12, n.º 6, págs. 1361-1367, dic. de 2020, ISSN: 1867-2469. DOI: [10.1007/s12551-020-00770-w](https://doi.org/10.1007/s12551-020-00770-w).
- [3] S. Joshi, V. Kumar, V. Venkataraman y K. C S, «A Review on Neural Networks and its Applications,» *Journal of Computer Technology & Applications*, vol. 14, pág. 2023, sep. de 2023. DOI: [10.37591/jocta.v14i2.1062](https://doi.org/10.37591/jocta.v14i2.1062).
- [4] V. Maggioni, C. Azevedo-Coste, S. Durand y F. Bailly, «Optimisation and Comparison of Markerless and Marker-Based Motion Capture Methods for Hand and Finger Movement Analysis,» *Sensors*, vol. 25, n.º 4, 2025, ISSN: 1424-8220. DOI: [10.3390/s25041079](https://doi.org/10.3390/s25041079).
- [5] A. Harrison, A. Jester, S. Mouli, A. Fratini y A. Jabran, «Systematic Evaluation of IMU Sensors for Application in Smart Glove System for Remote Monitoring of Hand Differences,» *Sensors*, vol. 25, n.º 1, 2025, ISSN: 1424-8220. DOI: [10.3390/s25010002](https://doi.org/10.3390/s25010002).
- [6] T. Unger, R. de Sousa Ribeiro, M. Mokni et al., «Upper limb movement quality measures: comparing IMUs and optical motion capture in stroke patients performing a drinking task,» *Frontiers in Digital Health*, vol. Volume 6 - 2024, 2024, ISSN: 2673-253X. DOI: [10.3389/fdgth.2024.1359776](https://doi.org/10.3389/fdgth.2024.1359776).
- [7] M. Hioki y H. Kawasaki, «Estimation of Finger Joint Angles from sEMG Using a Neural Network Including Time Delay Factor and Recurrent Structure,» *ISRN Rehabilitation*, vol. 2012, 2012. DOI: [10.5402/2012/604314](https://doi.org/10.5402/2012/604314).
- [8] N. Shrirao, N. P. Reddy y D. P. Kosuri, «Neural network committees for finger joint angle estimation from surface EMG signals,» *BioMedical Engineering OnLine*, vol. 8, n.º 1, pág. 2, 2009. DOI: [10.1186/1475-925X-8-2](https://doi.org/10.1186/1475-925X-8-2).

- [9] J. G. Ngeo, T. Tamei y T. Shibata, «Continuous and simultaneous estimation of finger kinematics using inputs from an EMG-to-muscle activation model,» *Journal of NeuroEngineering and Rehabilitation*, vol. 11, n.º 1, pág. 122, 2014. DOI: [10.1186/1743-0003-11-122](https://doi.org/10.1186/1743-0003-11-122).
- [10] H. Lee, D.-H. Kim e Y.-L. Park, «Explainable deep learning model for EMG-based finger angle estimation using attention,» *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 30, págs. 1877-1886, 2022. DOI: [10.1109/TNSRE.2022.3188275](https://doi.org/10.1109/TNSRE.2022.3188275).
- [11] C. Avian, S. W. Prakosa, M. Faisal y J.-S. Leu, «Estimating finger joint angles on surface EMG using manifold learning and long short-term memory with attention mechanism,» *Biomedical Signal Processing and Control*, vol. 71, pág. 103099, 2022. DOI: [10.1016/j.bspc.2021.103099](https://doi.org/10.1016/j.bspc.2021.103099).
- [12] J. Fan, L. Vargas, D. G. Kamper y X. Hu, «Robust neural decoding for dexterous control of robotic hand kinematics,» *Computers in Biology and Medicine*, vol. 162, pág. 107139, 2023. DOI: [10.1016/j.combiomed.2023.107139](https://doi.org/10.1016/j.combiomed.2023.107139).
- [13] N. Jiang, S. Muceli, B. Graimann y D. Farina, «Effect of arm position on the prediction of kinematics from EMG in amputees,» en, *Med. Biol. Eng. Comput.*, vol. 51, n.º 1-2, págs. 143-151, feb. de 2013.
- [14] S. Stapornchaisit, Y. Kim, A. Takagi, N. Yoshimura e Y. Koike, «Finger Angle Estimation From Array EMG System Using Linear Regression Model With Independent Component Analysis,» *Frontiers in Neurorobotics*, vol. 13, pág. 75, 2019. DOI: [10.3389/fnbot.2019.00075](https://doi.org/10.3389/fnbot.2019.00075).
- [15] BrainVision, *BrainAmp Product Page*, <https://brainvision.com/products/brainamp-standard/>, [Accessed 21-05-2025], 2023.
- [16] Technomed, *Disposable Adhesive Electrodes Product Page*, <https://technomedmedical.com/product/technomed-disposable-adhesive-4-disk-electrodes/>, [Accessed 21-05-2025].
- [17] BrainVision, *Brain Connect Product Page*, <https://brainvision.com/products/usb-2-adapter-bua/>, [Accessed 21-05-2025], 2021.
- [18] Arduino Documentation, [Accessed 10-06-2025], 2025. dirección: <https://docs.arduino.cc/>.
- [19] BrainVision Recorder | Brain Products GmbH >Solutions — brainproducts.com, <https://www.brainproducts.com/solutions/recorder/>, [Accessed 12-06-2025].
- [20] BrainVision Analyzer | Brain Products GmbH >Solutions — brainproducts.com, <https://www.brainproducts.com/solutions/analyzer/>, [Accessed 12-06-2025].
- [21] Conda Documentation; conda 25.3.1 documentation — docs.conda.io, <https://docs.conda.io/projects/conda/en/stable/>, [Accessed 21-05-2025].
- [22] Welcome to Planemo's documentation!; Planemo 0.75.31.dev0 documentation — planemo.readthedocs.io, <https://planemo.readthedocs.io/>, [Accessed 21-05-2025].

- [23] S. R. Krishnan y C. S. Seelamantula, «On the Selection of Optimum Savitzky-Golay Filters,» *IEEE Transactions on Signal Processing*, vol. 61, n.º 2, págs. 380-391, 2013. DOI: [10.1109/TSP.2012.2225055](https://doi.org/10.1109/TSP.2012.2225055).
- [24] Google, *Mediapipe Documentation*, https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker?hl=es-419, [Accessed 21-05-2025].
- [25] M. Schuenke, *Thieme atlas of anatomy*, L. M. Ross y E. D. Lamperti, eds. New York, NY: Thieme Medical, jul. de 2005.
- [26] J. Hochschild, *Functional anatomy for physical therapists*, en. Stuttgart, Germany: Thieme Publishing Group, nov. de 2015.

A Entorno de Conda y dependencias de Python

```
pip freeze
```

```
absl-py==2.3.1
astunparse==1.6.3
attrs==25.3.0
certifi==2025.7.14
cffi==1.17.1
charset-normalizer==3.4.2
contourpy==1.3.2
cycler==0.12.1
flatbuffers==25.2.10
fonttools==4.59.0
gast==0.6.0
google-pasta==0.2.0
grpcio==1.73.1
h5py==3.14.0
idna==3.10
jax==0.6.2
jaxlib==0.6.2
keras==3.10.0
kiwisolver==1.4.8
libclang==18.1.1
Markdown==3.8.2
markdown-it-py==3.0.0
MarkupSafe==3.0.2
matplotlib==3.10.3
mdurl==0.1.2
mediapipe==0.10.21
ml_dtypes==0.5.1
namex==0.1.0
numpy==1.26.4
opencv-contrib-python==4.11.0.86
opencv-python==4.8.1.78
opt_einsum==3.4.0
optree==0.16.0
packaging==25.0
pandas==2.3.1
pillow==11.3.0
protobuf==4.25.8
pycparser==2.22
Pygments==2.19.2
pyparsing==3.2.3
python-dateutil==2.9.0.post0
pytz==2025.2
requests==2.32.4
rich==14.0.0
scipy==1.15.3
sentencepiece==0.2.0
six==1.17.0
sounddevice==0.5.2
tensorboard==2.19.0
tensorboard-data-server==0.7.2
tensorflow==2.19.0
tensorflow-io-gcs-filesystem==0.37.1
termcolor==3.1.0
typing_extensions==4.14.1
tzdata==2025.2
urllib3==2.5.0
Werkzeug==3.1.3
wrapt==1.17.2
```

B Tutorial para usar BrainVision Recorder

En este anexo se muestra el proceso completo que se ha llevado a cabo en este PFG para la grabación de señales EMG con BrainVision Recorder.

Al abrir la aplicación de BrainVision Recorder nos recibe la siguiente pestaña que se puede ver en la Figura B.1. Se marca con un uno rojo (1) el menú al que accederemos para definir los ajustes de la grabación. Se marca con un dos verde (2) los botones que afectan al visor de la señal.

Empezaremos accediendo a `File` para crear un nuevo entorno con sus ajustes y parametros (`New Workspace...`) o podremos abrir un entorno preconfigurado (`Open Worskpace...`) como se ve en la Figura B.2.

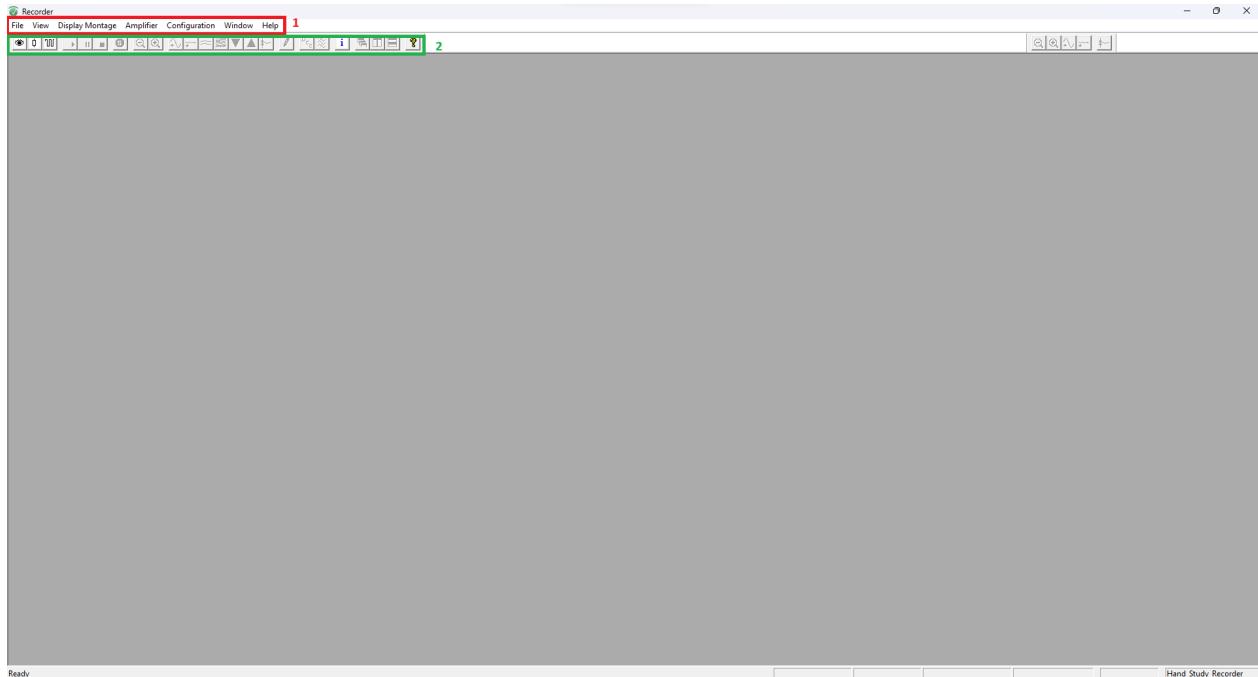


Figura B.1: Recorder



Figura B.2: Recorder

B.1 Creación del entorno de trabajo

Si se posee un entorno preparado puede importarse y saltar directamente a la Sección B.2. Si en cambio, se decide crear un nuevo entorno, estas son las instrucciones a seguir y los parámetros a definir:

1. Definir la ruta en la que se guardarán los archivos y la nomenclatura de los mismos. (Figura B.3).
2. Buscar el amplificador (asegurar que está encendido desde el interruptor de detrás) y definir los parámetros como aparecen en la Figura B.4.
3. Los filtros de guardado ([Raw Data Saving Filters](#)) y segmentación ([Segmentation Filters](#)) se deberán dejar vacíos. (Figura B.5).
4. Para la visualización ([Display Filters](#)) si que se definen filtros para eliminar la influencia de la red eléctrica y poder apreciar la señal. Los parámetros usados se recogen en la Figura B.6 y a continuación:
 - Number of Channels: 10
 - Sampling Rate [Hz]: 2500

- Low Cutoff [s]: 10
 - High Cutoff [Hz]: 1000
 - Notch Filter [Hz]: 50
5. Por último, la segmentación (Segmentation/Averaging) se deja como está por defecto, vacía. (Figura B.7).

B.2 Creación del *Display Montage*

También necesitaremos definir un *Display Montage* para establecer aquellos electrodos que están emparejados para el estudio de EMG bipolar. De nuevo, se puede definir uno nuevo, o se puede importar uno ya preparado. Para ello accedemos a *Display Montage* y seleccionamos *New...* o *Edit...* según lo que queramos (Figura B.8). Si tu caso es el último entonces puedes saltar a la Sección B.3.

B.3 Comprobar impedancias y grabar

Para acceder a la pestaña de impedancia se debe pulsar en el icono marcado con un recuadro rojo en la Figura B.11. Desde ahí se deberá ajustar los electrodos para reducir lo máximo posible la impedancia. Desde las opciones a la derecha (marcado en verde en la Figura B.11) también se deberá comprobar la impedancia de los electrodos de referencia y tierra. En la imagen se ve una cabeza, debido a que el software está preparado para el análisis de electroencefalografía y permitiría situar los electrodos en su posición sobre el cuero cabelludo para mapear la actividad cerebral en un modelo.

Conseguido esto, al pulsar en el icono del visor (marcado en rojo en la Figura B.12) veremos lo que se ve en la misma Figura B.12. Para grabar solo deberemos pulsar el botón marcado en verde. Una vez grabando se encenderá el piloto de la parte inferior (**SAVING HDD**) y podremos pulsar el botón marcado en azul para detener la grabación.

En la franja naranja veremos aparecer las señales que mandemos a través del USB 2 Adapter de BrainVision en forma de *triggers*.

1. Se elige formato bipolar. (Figura B.9).
2. Se selecciona con los menús desplegables los electrodos que están enfrentados. (Figura B.10).

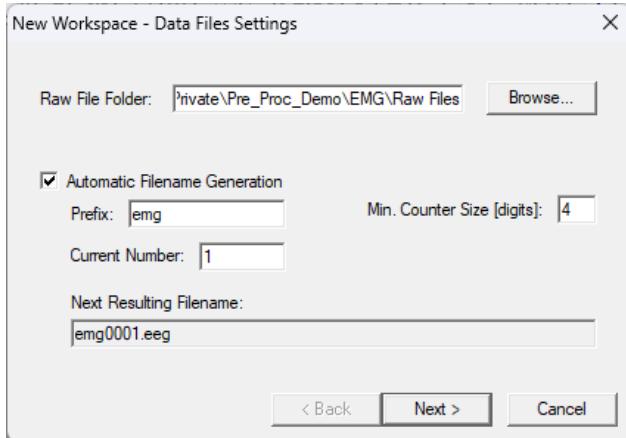


Figura B.3: New Workspace - Data File Settings

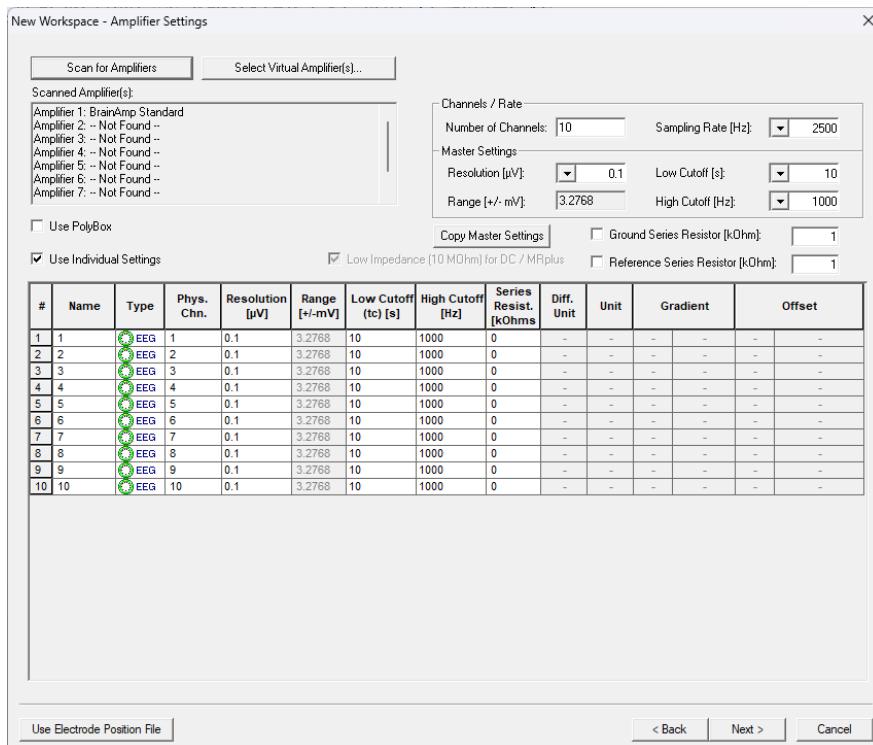


Figura B.4: New Workspace - Amplifier Settings

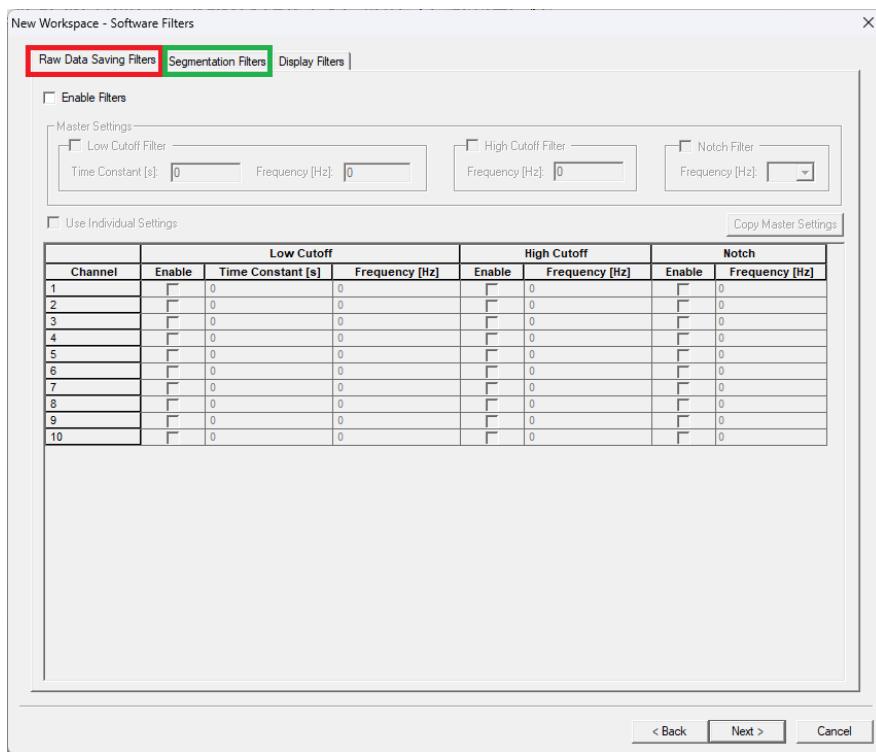


Figura B.5: New Workspace - Software Filters

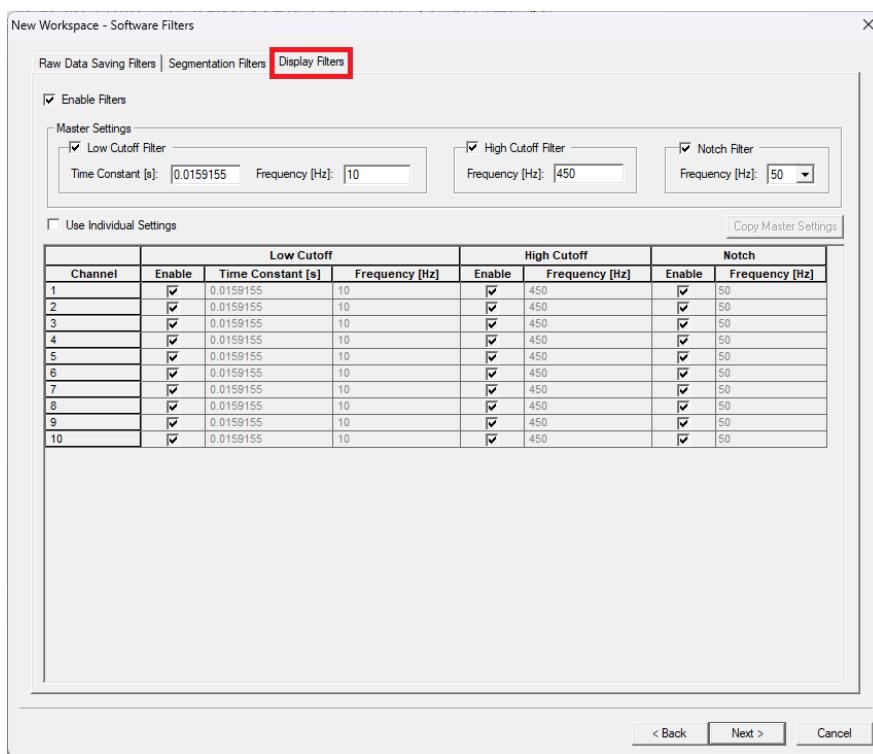


Figura B.6: New Workspace - Software Filters

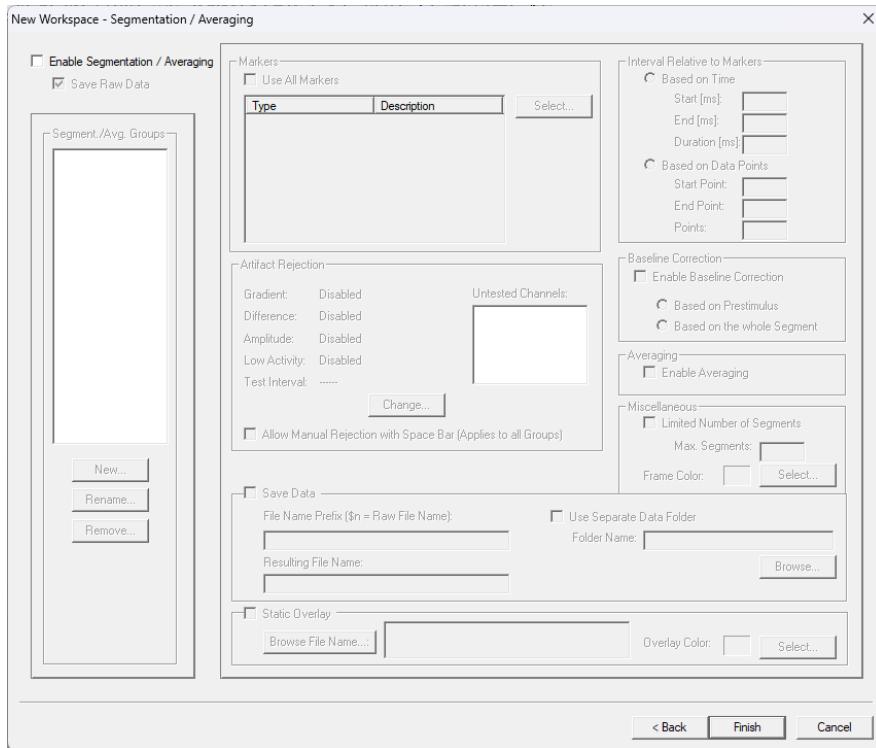


Figura B.7: New Workspace - Segmentation / Averaging

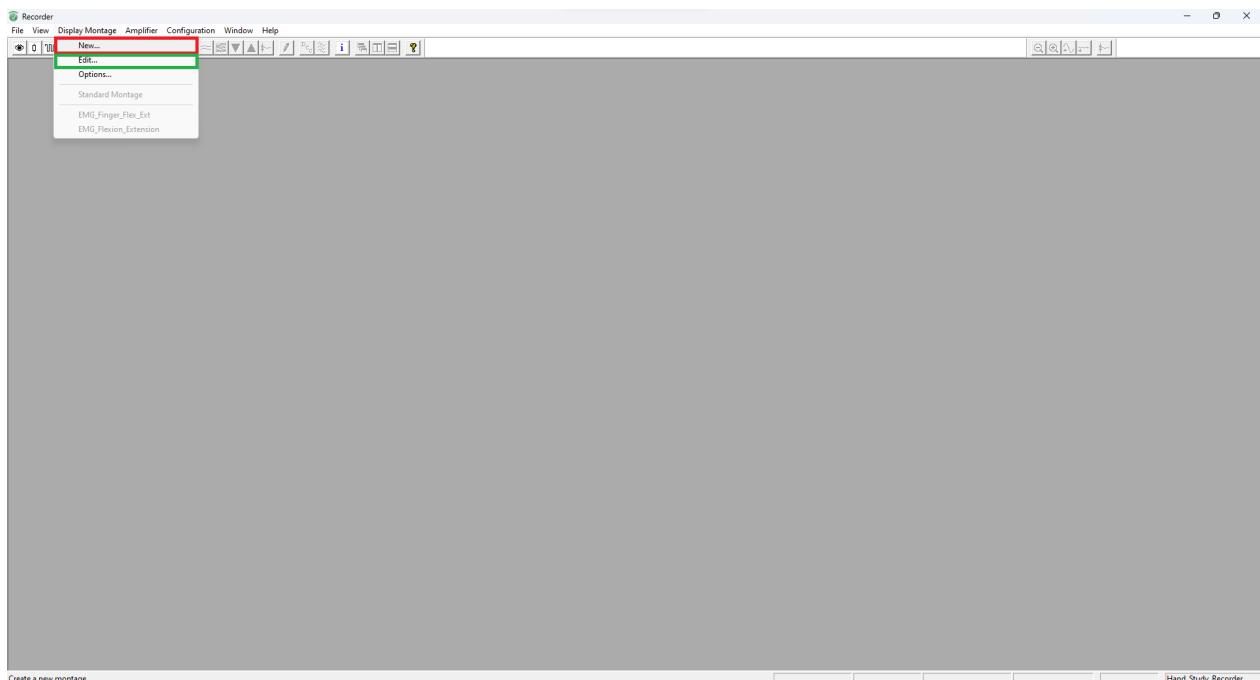


Figura B.8: Recorder

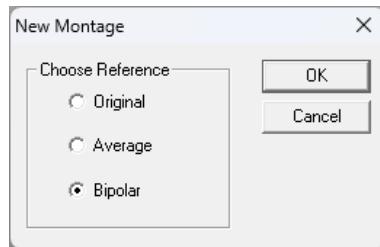


Figura B.9: New Montage

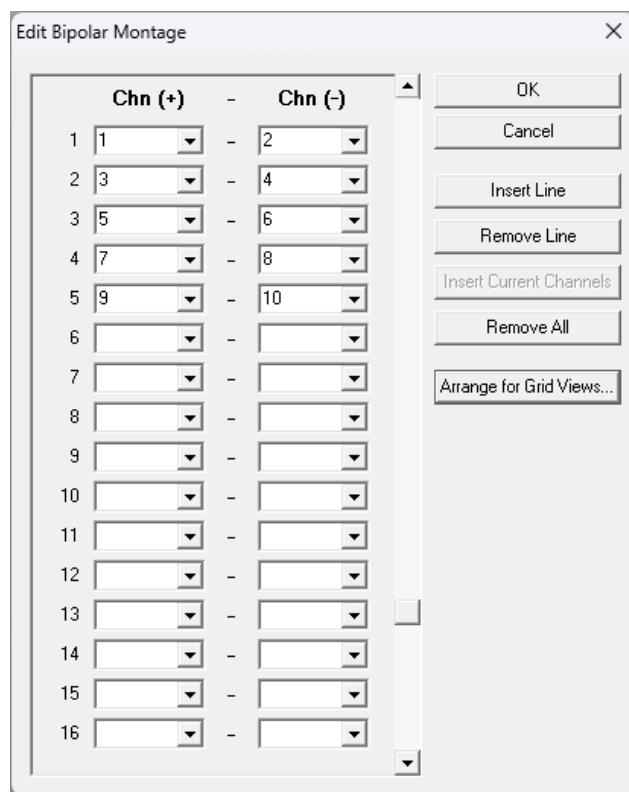


Figura B.10: Edit Bipolar Montage

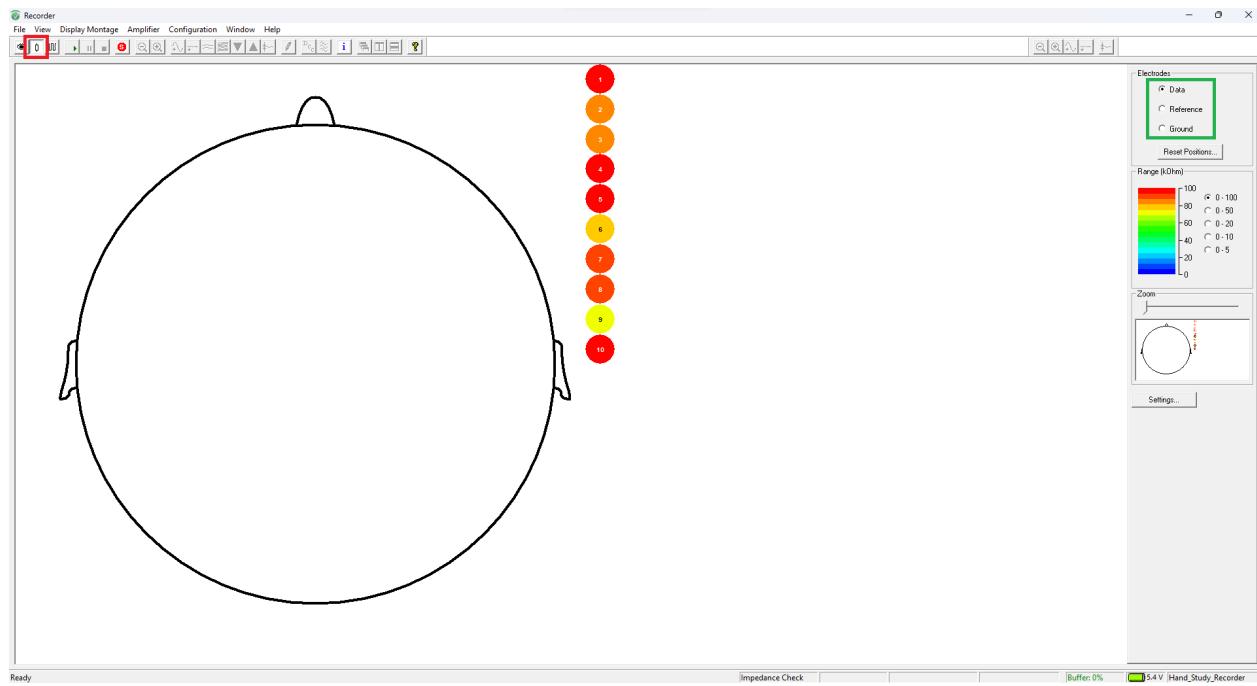


Figura B.11: Recorder - Impedance Check

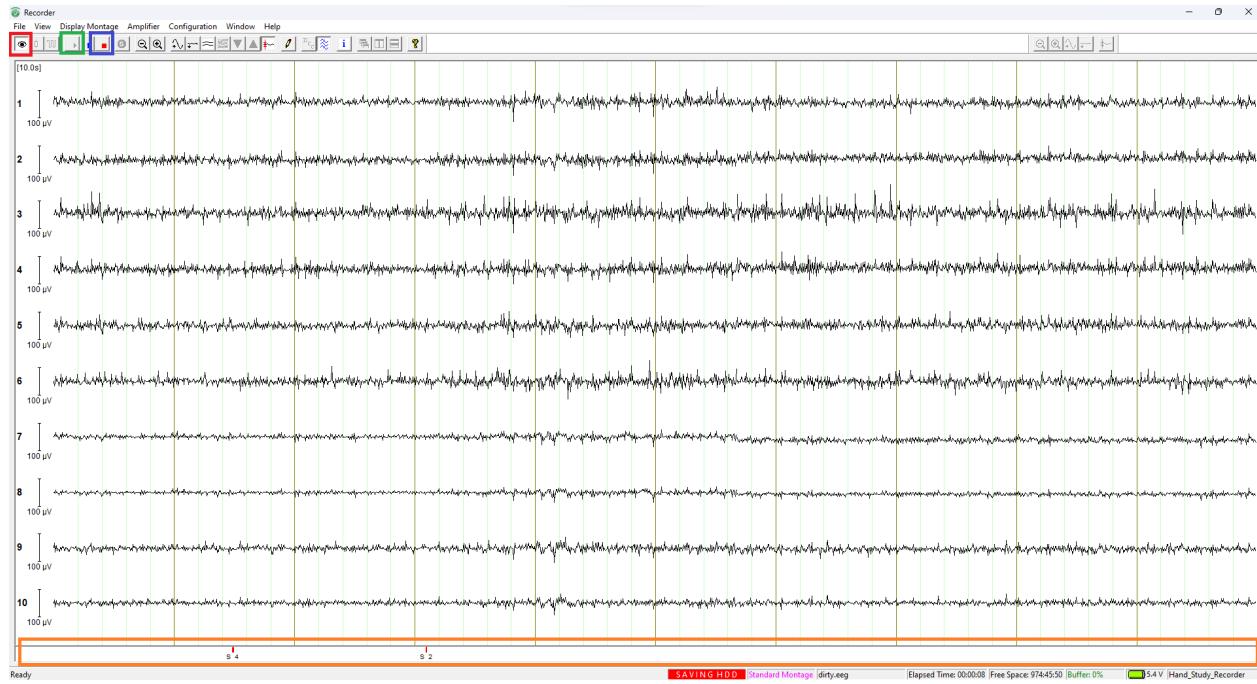


Figura B.12: Recorder - Visor

C Tutorial para usar BrainVision Analyzer

En este anexo se muestra el proceso completo que se ha llevado a cabo en este PFG para el procesado de las señales EMG para tanto el entrenamiento del modelo de ML como para el análisis llevado a cabo. Todo ello con BrainVision Analyzer y Python.

Al abrir la aplicación de BrainVision Analyzer se nos presenta con la siguiente ventana (Figura C.1). La franja marcada en rojo será donde más adelante podremos ver nuestra área de trabajo con nuestros archivos de EMG. En verde podremos ver la barra de selección de menús.

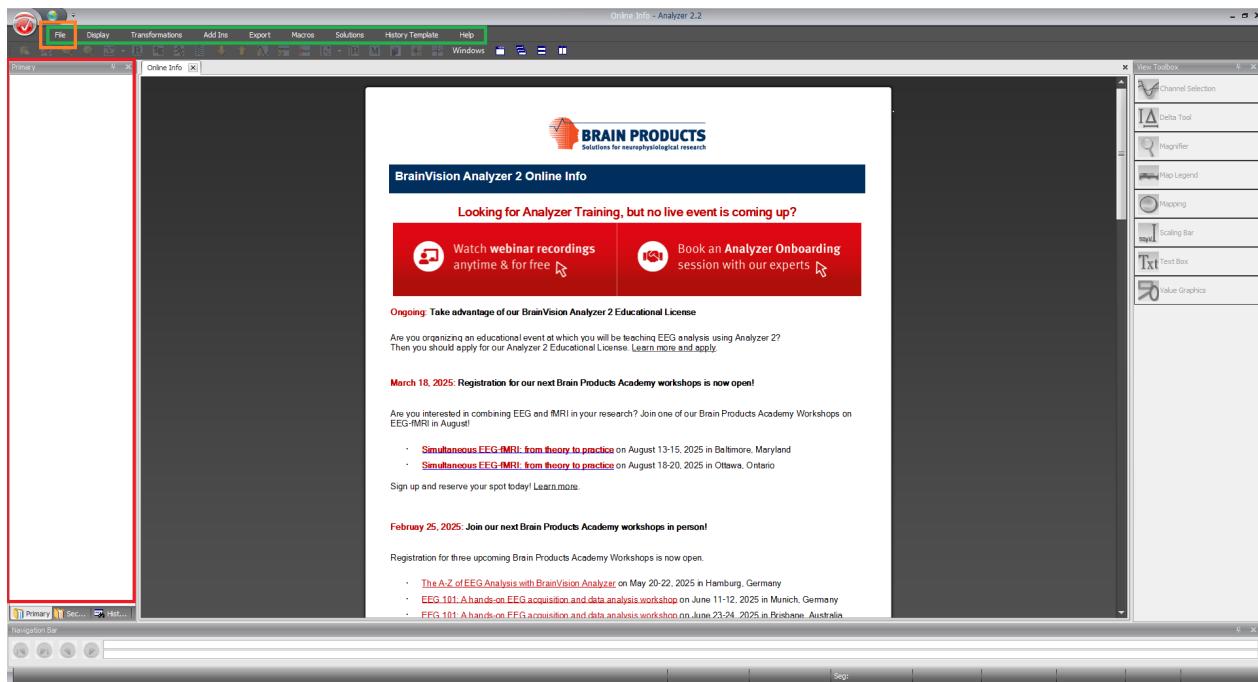


Figura C.1: Analyzer

C.1 Creación del entorno de trabajo

Desde el menú de File (marcado en naranja en la Figura C.1 y representado en la Figura C.2) podremos importar (Open) o crear (New) un nuevo entorno de trabajo. Si ya se dispone de un entorno preparado con anterioridad, se puede saltar a la Sección C.2. Sin embargo, la creación consiste únicamente en definir la ubicación de las carpetas de trabajo, así que es necesario hacerlo si se ha cambiado de ubicación aunque se disponga de un entorno anterior.

Para crear el entorno deberemos definir las carpetas en las que se encuentran nuestros archivos y en las que se guardarán. Dichas carpetas son las siguientes (Figura C.3):

- Raw Files: ubicación de los archivos de EMG obtenidos.
- History Files: ubicación para los archivos de historia que se generan durante el procesado de la señal.
- Export Files: ubicación en la que se guardarán las señales procesadas que se exporten.

Si todo se ha definido correctamente deberían aparecer nuestras señales grabadas en el área roja de la Figura C.1. En ese menú seleccionamos la carpeta con la señal que deseamos y pulsando sobre Raw Data se debería obtener lo que se puede ver la Figura C.4.

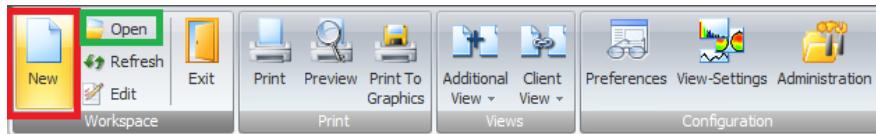


Figura C.2: Menu - File

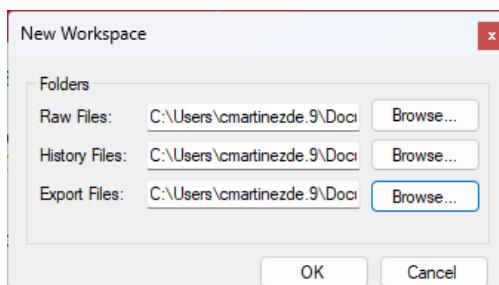


Figura C.3: New Workspace

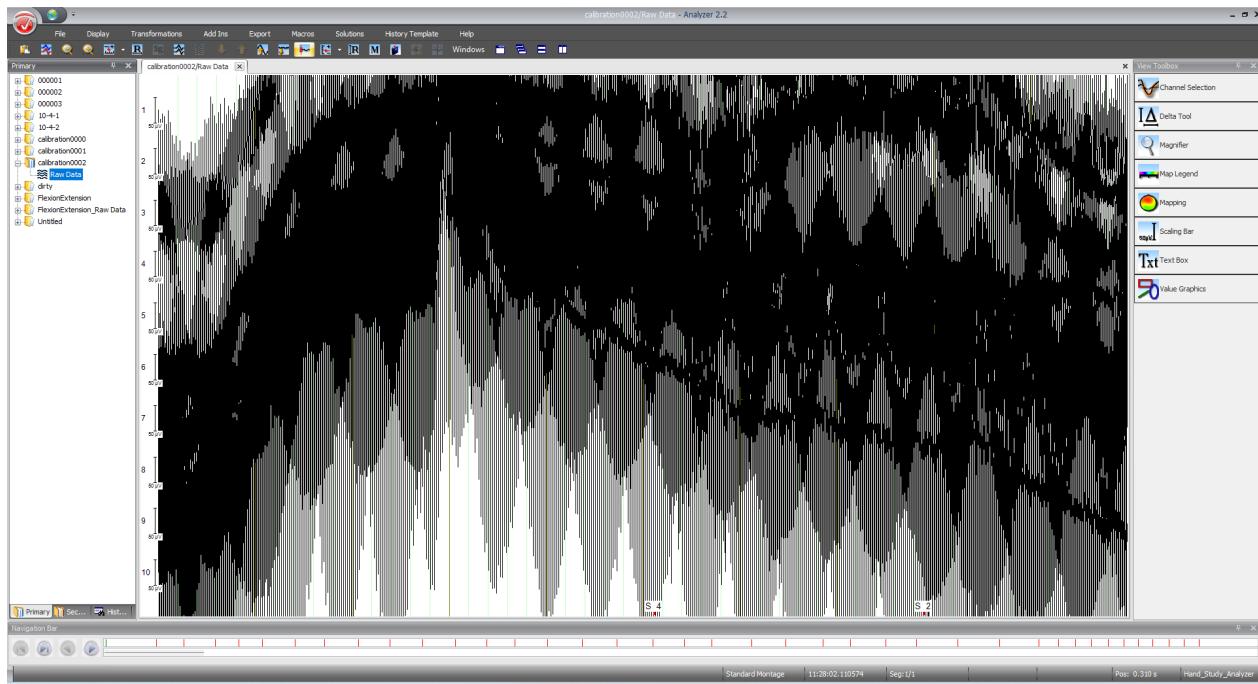


Figura C.4: Analyzer - Display

C.2 Procesado de la señal

En esta sección se va a describir todo el proceso de la señal de EMG. Este proceso se puede diferenciar en dos partes, una primera que hace referencia al preprocesado que se usará para la creación del set de datos con el que se entrenará al modelo de ML y luego el procesado realizado para analizar la relación de las señales y su relevancia estadística.

En orden se hace lo siguiente:

1. Dentro del menú Transformations se abre la pestaña de filtros IIR. (Figura C.5).
2. Se define el filtro con los valores que se veían en [13]. (Figura C.6).
 - Low Cutoff [s]: 10
 - High Cutoff [Hz]: 1000
 - Notch Filter [Hz]: 50
3. De nuevo aplicamos una transformación, esta vez para cambiar la frecuencia de muestreo (*sampling rate*). (Figura C.7).

4. Dentro de la nueva pestaña cambiamos la frecuencia a 1024 Hz. (Figura C.8).
5. Aplicamos una nueva transformación para segmentar la señal. (Figura C.9).

En este paso es donde se separan los procedimientos para entrenar al modelo y para el análisis. En el primer caso se realiza una única segmentación para eliminar las partes del EMG a las que no corresponde un set de coordenadas (debido a que la cámara todavía no había grabado). En el segundo separaremos entre movimientos de apertura y cierre de la mano para el estudio de la diferencia que existe.

6. Seleccionamos que el nuevo segmento sea definido por un marcador inicial y final. (Figura C.10)
7. Seleccionamos el marcador S4 (cable verde) como inicio y S2 (cable rojo) como fin. (Figura C.11).
8. Terminamos de segmentar sin crear un set de datos nuevo. (Figura C.12).
9. Desde el menú de transformaciones vamos a acceder a la herramienta de edición de canales. (Figura C.13).
10. Buscamos enfrentar los canales individuales de cada electrodo para obtener un estudio bipolar. Agrupamos los canales según la posición elegida de los electrodos, para el estudio se usa lo que se ve en la Figura C.14.
11. Buscamos en el menú de transformaciones el apartado de FFT (Fast Fourier Transform). (Figura C.15).
12. Dentro de la herramienta de FFT no necesitamos modificar nada, por lo que aceptamos. (Figura C.16).
13. Podemos ver las bandas desde el visor. Desde el ícono marcado en rojo en la Figura C.17 podremos disminuir el rango para ver la imagen con más claridad.
14. Para hacer el estudio más homogéneo usamos la herramienta de promedio (*average*) para suavizar las señales. (Figura C.18).

Con esto habría terminado el procesado de la señal para el estudio. Ahora exportaremos a un formato que nos permita trabajar en Python. Para ello exportaremos los datos en formato binario.

15. Dentro del apartado para exportar dentro del menú superior podemos exportar como datos genéricos. (Figura C.19).
 16. Para exportar, aunque no sea estrictamente necesario, sí que es interesante marcar las casillas para generar un fichero de encabezados (con información sobre el experimento) y de marcadores (con información sobre el instante de detección de los marcadores) todo ello en formato XML.
-

(Figura C.20).

17. Ahora es importante marcar que los datos se expresen en formato binario y orientados de forma vectorizada. El script de Python empleado ya está preparado para tratar con este tipo de datos. (Figura C.21).
18. Seleccionamos que el tipo de formato de los puntos sea de coma flotante de 32 bits según el estándar de la IEEE (*Institute of Electrical and Electronics Engineers*). (Figura C.22).
19. Nos aseguramos de que exportamos todos los canales. Como se ve en la Figura C.23 solo los canales impares aparecen, esto es debido a la edición hecha unos pasos atrás para darle una forma bipolar a nuestras lecturas.

Y con esto termina el procesado y exportado de datos con BrainVision Analyzer. Despues podremos importarlos desde Python y, o bien, representarlos con ayuda de librerías como matplotlib, o bien, realizar un filtrado de datos para almacenarlos en la base de datos que más tarde proporcionaremos al modelo de aprendizaje automático.

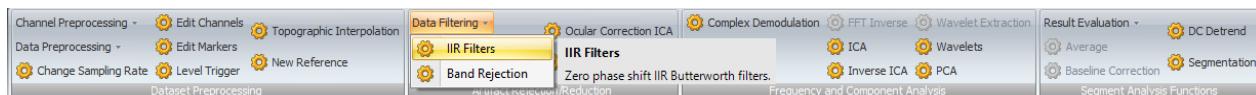


Figura C.5: Menu > Transformations

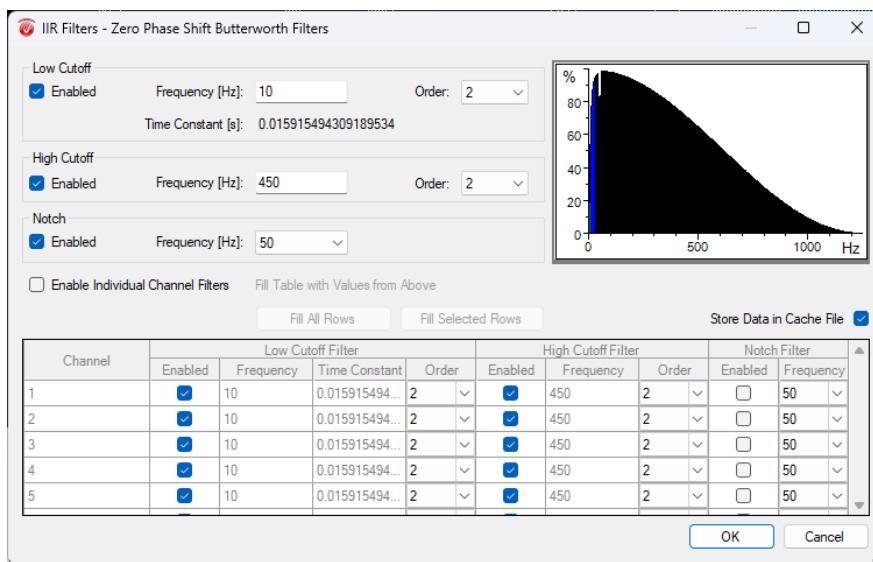


Figura C.6: IIR Filters

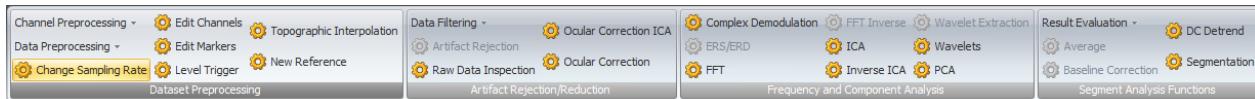


Figura C.7: Menu > Transformations

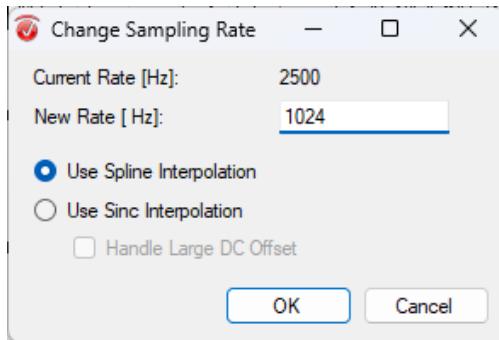


Figura C.8: Change Sampling Rate

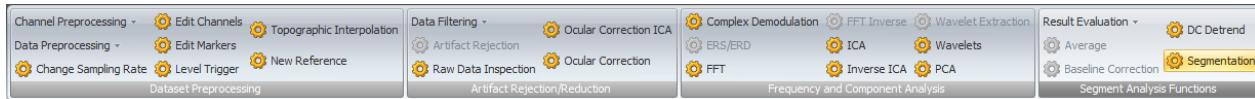


Figura C.9: Menu > Transformations

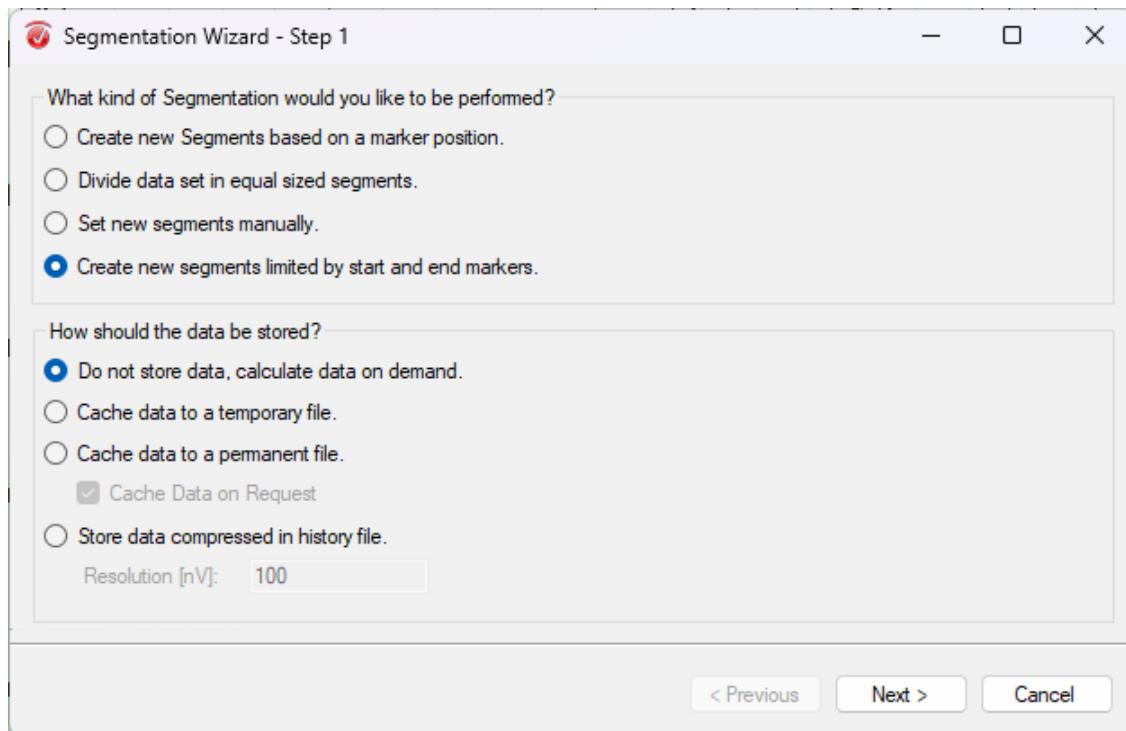


Figura C.10: Segmentation Wizard - Step 1

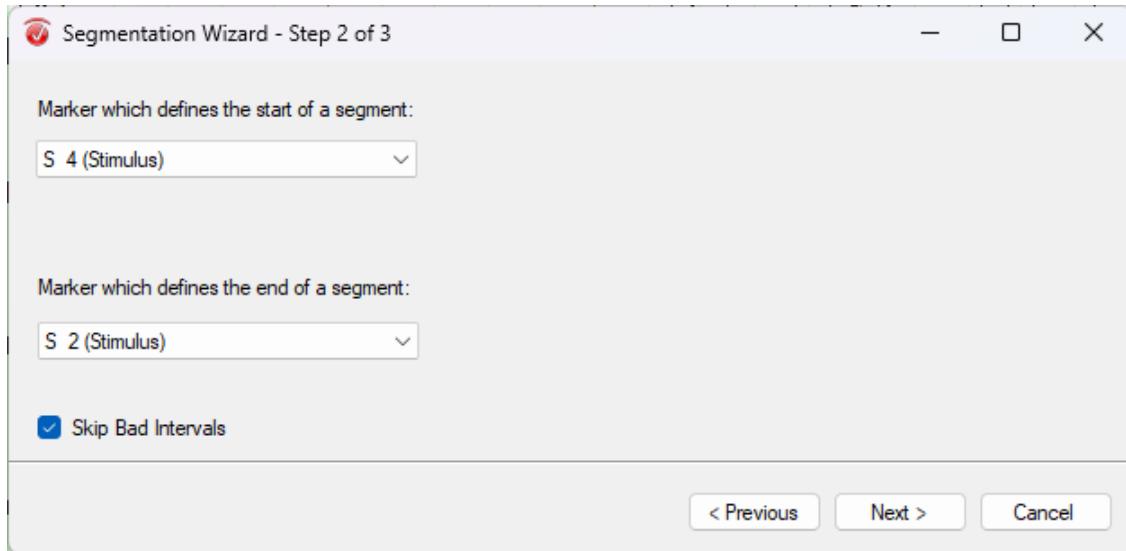


Figura C.11: Segmentation Wizard - Step 2

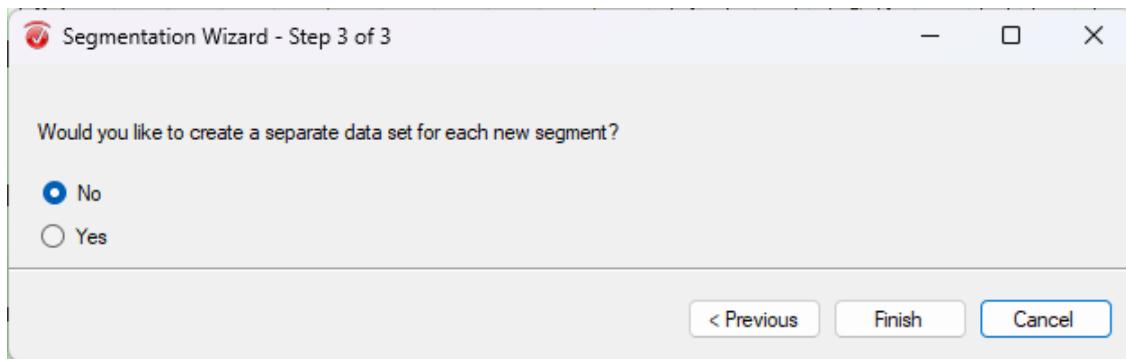


Figura C.12: Segmentation Wizard - Step 3

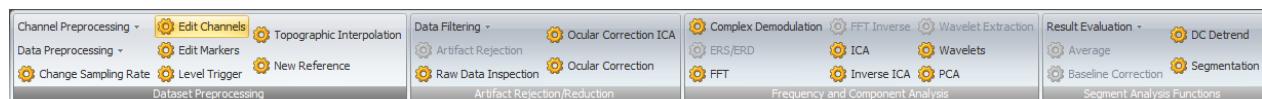


Figura C.13: Menu > Transformations

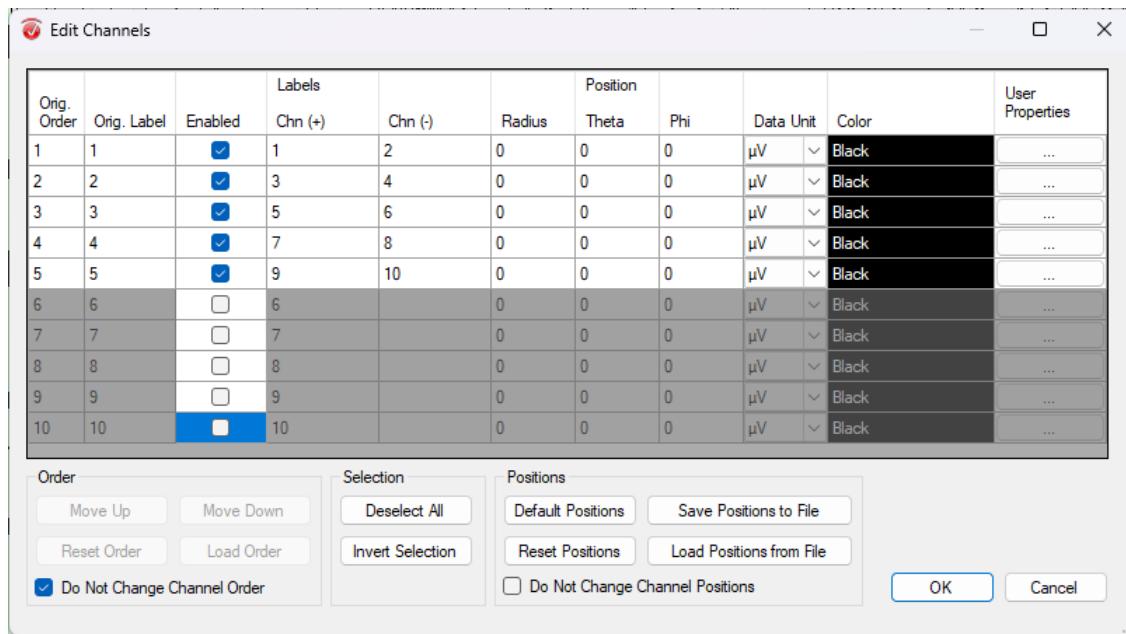


Figura C.14: Edit Channels



Figura C.15: Menu > Transformations

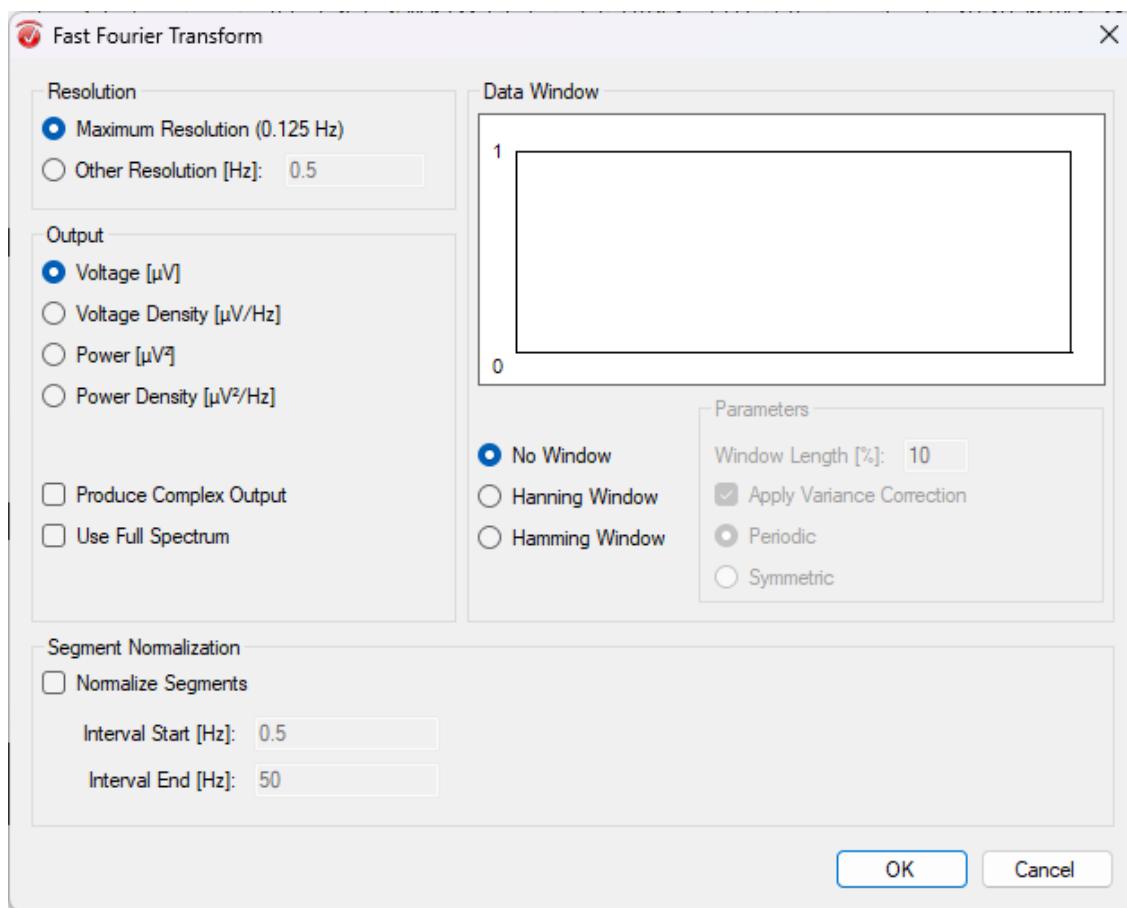


Figura C.16: Fast Fourier Transform

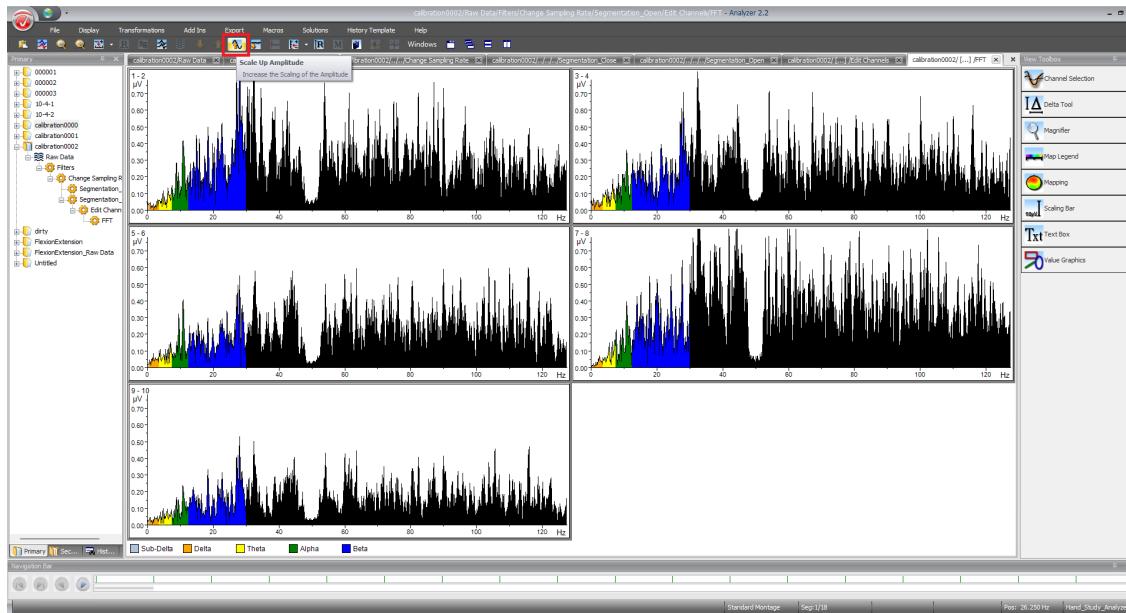


Figura C.17: Analyzer - Visor

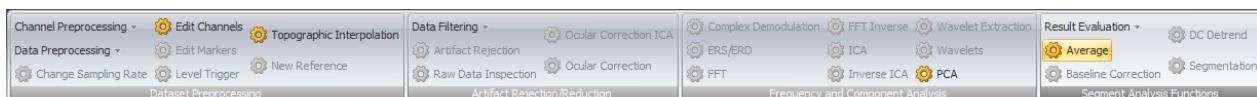


Figura C.18: Menu > Transformations

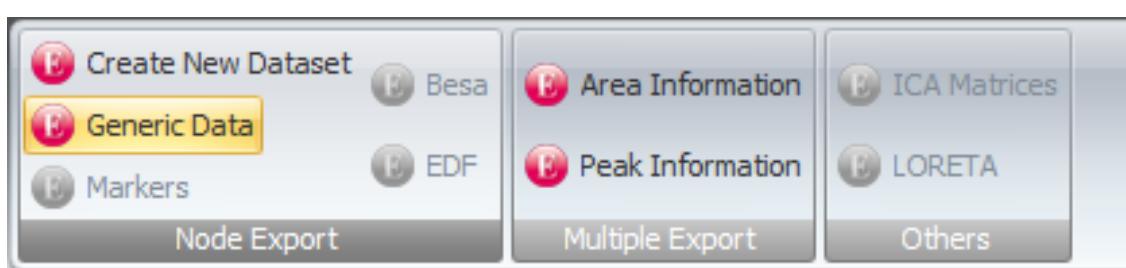


Figura C.19: Menu > Export

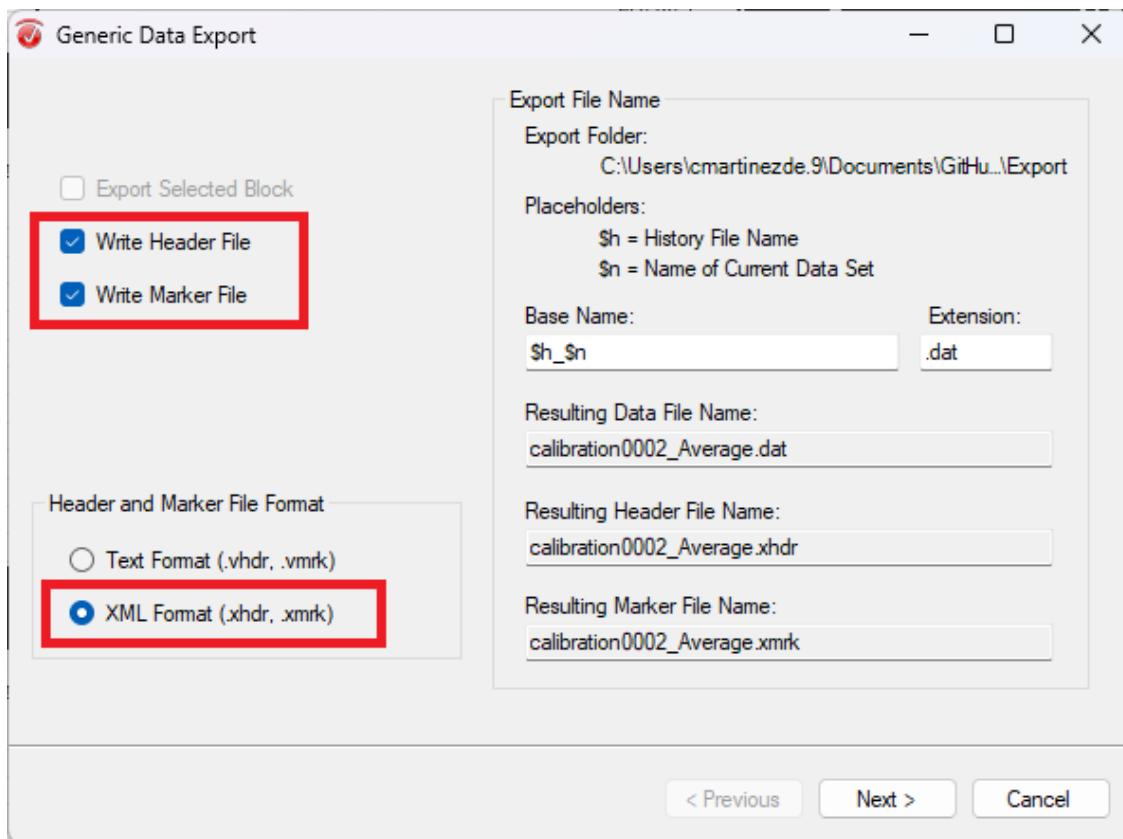


Figura C.20: Generic Data Export

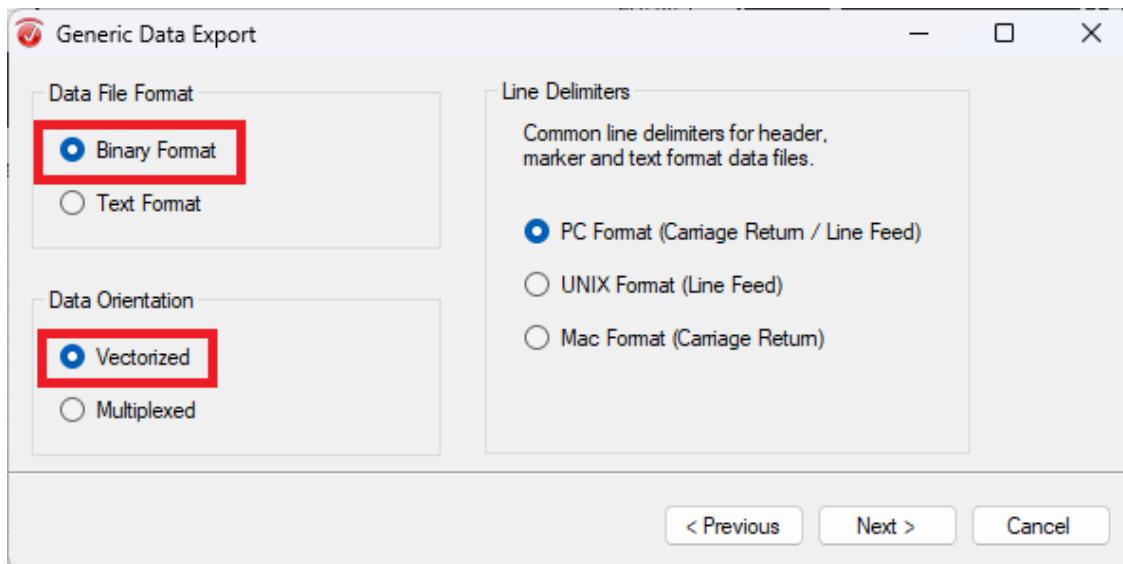


Figura C.21: Generic Data Export

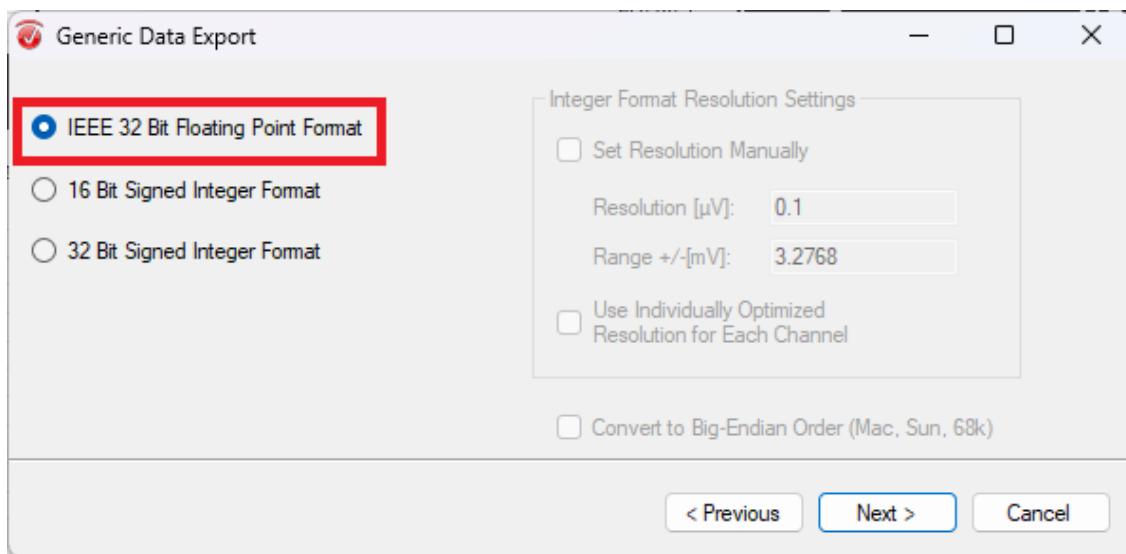


Figura C.22: Generic Data Export

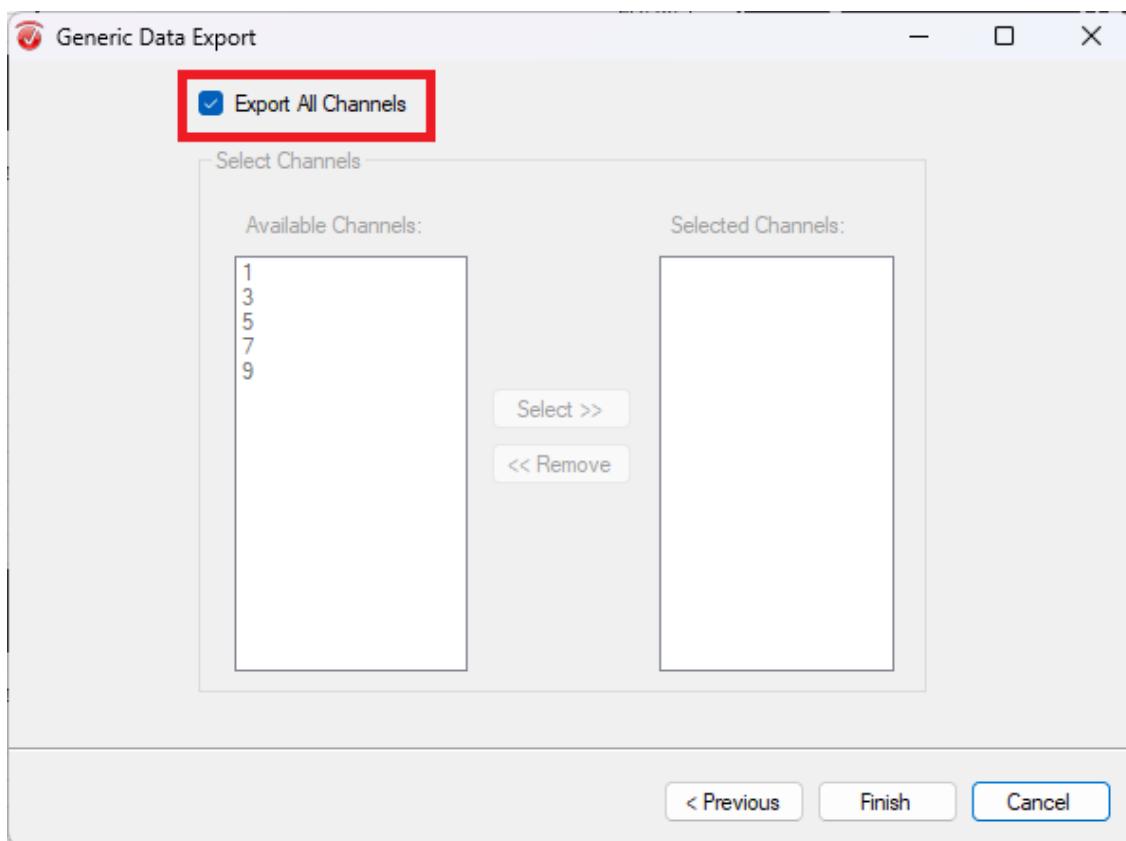


Figura C.23: Generic Data Export

D Código y funciones de Python

D.1 Funciones y script para grabar

Función para obtener el nombre del siguiente vídeo:

Python

```
1 def get_next_filename(folder="video_data", prefix="", extension=".mp4", digits=6):
2     os.makedirs(folder, exist_ok=True)
3     i = 1
4     while True:
5         filename = f"{prefix}{i:0{digits}d}{extension}"
6         full_path = os.path.join(folder, filename)
7         if not os.path.exists(full_path):
8             return full_path
9         i += 1
```

Función para obtener imagen:

Python

```
1 def get_video():
2     frame, _ = freenect.sync_get_video()
3     return cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
```

Script para toma de imágenes:

Python

```
1 arduino = serial.Serial("/dev/ttyACM0", 9600, timeout=1)
2 time.sleep(2)
3
4 output_filename = get_next_filename()
5 fourcc = cv2.VideoWriter_fourcc(*"mp4v")
6 output_video = cv2.VideoWriter(output_filename, fourcc, 32.0, (640, 480))
7
8 frame = get_video()
9 cv2.imshow("Kinect Video", frame)
10
11 arduino.write(b"H")
```

```
12
13 while True:
14     frame = get_video()
15     output_video.write(frame)
16     cv2.imshow("Kinect Video", frame)
17
18     # exit with q
19     if cv2.waitKey(1) & 0xFF == ord("q"):
20         arduino.write(b'L')
21         break
22
23 output_video.release()
24 cv2.destroyAllWindows()
25 arduino.close()
```

D.2 Funciones relacionadas con visión artificial

Función para aplicar el modelo de visión artificial y obtener las coordenadas:

Python

```
1 def image_recognition_get_csv(file_path):
2     if not os.path.exists(file_path):
3         raise FileNotFoundError(f"El archivo de video {file_path} no existe.")
4
5     MODEL_PATH = "./hand_landmarker.task"
6
7     # initialize image recognition model
8     base_options = python.BaseOptions(model_asset_path=MODEL_PATH)
9     options = vision.HandLandmarkerOptions(
10         base_options=base_options, running_mode=vision.RunningMode.VIDEO, num_hands=1
11     )
12     landmarker = vision.HandLandmarker.create_from_options(options)
13
14     # import video and properties
15     video_data = cv2.VideoCapture(file_path)
16     fps = int(video_data.get(cv2.CAP_PROP_FPS))
17     width = int(video_data.get(cv2.CAP_PROP_FRAME_WIDTH))
18     height = int(video_data.get(cv2.CAP_PROP_FRAME_HEIGHT))
19     output_path = file_path + "_processed.mp4"
20     fourcc = cv2.VideoWriter_fourcc(*"mp4v")
21     out = cv2.VideoWriter(output_path, fourcc, fps, (width, height))
22
23     coord_data = []
24     frame_id = 0
25
26     while video_data.isOpened():
27         ret, frame = video_data.read()
28         if not ret:
29             break
```

```

30
31     timestamp_ms = int((frame_id / fps) * 1000)
32     frame_id += 1
33
34     frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
35     mp_image = mp.Image(image_format=mp.ImageFormat.SRGB, data=frame_rgb)
36     result = landmarker.detect_for_video(mp_image, timestamp_ms)
37
38     if result.hand_landmarks:
39         for hand_idx, hand_landmarks in enumerate(result.hand_landmarks):
40             for i, landmark in enumerate(hand_landmarks):
41                 coord_data.append(
42                     {
43                         "frame": frame_id,
44                         "hand_id": hand_idx,
45                         "landmark_id": i,
46                         "x": landmark.x,
47                         "y": landmark.y,
48                         "z": landmark.z,
49                     }
50                 )
51
52     # draw a video of model seen
53     mp.solutions.drawing_utils.draw_landmarks(
54         frame,
55         landmark_list_to_landmark_proto(hand_landmarks),
56         mp.solutions.hands.HAND_CONNECTIONS,
57     )
58
59     out.write(frame)
60
61     video_data.release()
62     out.release()
63     cv2.destroyAllWindows()
64
65     coord_dataframe = pd.DataFrame(coord_data)
66     csv_path = file_path + "processed.csv"
67     coord_dataframe.to_csv(csv_path, index=False)
68     # returns the csv too
69     return coord_dataframe

```

Función necesaria para dibujar sobre el *frame* el modelo obtenido:

Python

```

1 def landmark_list_to_landmark_proto(landmarks):
2     landmark_proto = landmark_pb2.NormalizedLandmarkList()
3     for lm in landmarks:
4         landmark_proto.landmark.add(x=lm.x, y=lm.y, z=lm.z)
5     return landmark_proto

```

D.3 Funciones relacionadas con el tratamiento del vídeo

Función para pivotar la tabla y convertir cada fila en un *frame*:

Python

```

1 def pivot_csv_table(video_data):
2     video_data_wide = video_data.pivot_table(
3         index="frame", columns="landmark_id", values=["x", "y", "z"]
4     )
5
6     # flatten multi-index columns
7     video_data_wide.columns = [
8         f"coord_{lm}{lm}" for coord, lm in video_data_wide.columns
9     ]
10    video_data_wide.reset_index(inplace=True)
11
12    return video_data_wide

```

Función para el suavizado de las coordenadas grabadas:

Python

```

1 def smooth_video_coord(video_data, WINDOW=33, ORDER=2):
2     # Make a copy of the original DataFrame
3     video_data_filtered = video_data.copy()
4
5     # Loop through all columns
6     for col in video_data.columns:
7         # Check if it's a coordinate column like x_lm#
8         if col.startswith(("x_lm", "y_lm", "z_lm")) and video_data[col].dtype != "O":
9             if len(video_data[col]) >= WINDOW:
10                 video_data_filtered[col] = savgol_filter(video_data[col], WINDOW, ORDER)
11             else:
12                 video_data_filtered[col] = video_data[col] # fallback if too short
13
14
15    return video_data_filtered

```

Función de obtención de los ángulos de flexión a partir de articulaciones:

Python

```

1 def get_hand_angles(video_data):
2     angle_columns = [
3         "frame",
4         "alfa_I",
5         "alfa_II",
6         "beta_I",
7         "beta_II",
8         "beta_III",
9         "gamma_I",
10        "gamma_II",
11        "gamma_III",
12        "lambda_I",

```

```
13     "lambda_II",
14     "lambda_III",
15     "epsilon_I",
16     "epsilon_II",
17     "epsilon_III",
18 ]
19
20 list_points = [
21     [1, 2, 3],    # alfa_I
22     [2, 3, 4],    # alfa_II
23     [5, 6, 7],    # beta_I
24     [6, 7, 8],    # beta_II
25     [0, 5, 6],    # beta_III
26     [9, 10, 11],   # gamma_I
27     [10, 11, 12],   # gamma_II
28     [0, 9, 10],   # gamma_III
29     [13, 14, 15],   # lambda_I
30     [14, 15, 16],   # lambda_II
31     [0, 13, 14],   # lambda_III
32     [17, 18, 19],   # epsilon_I
33     [18, 19, 20],   # epsilon_II
34     [0, 17, 18],   # epsilon_III
35 ]
36
37 max_frames = video_data["frame"].max()
38 angle_df = pd.DataFrame(columns=angle_columns)
39
40 for frame in range(max_frames):
41     frame_data = video_data[video_data["frame"] == frame + 1]
42     angle_df.at[frame, "frame"] = frame + 1
43
44     for i, point in enumerate(list_points):
45         P1 = np.array(
46             [
47                 frame_data[f"x_lm{point[0]}"],
48                 frame_data[f"y_lm{point[0]}"],
49                 frame_data[f"z_lm{point[0]}"],
50             ]
51         )
52         P2 = np.array(
53             [
54                 frame_data[f"x_lm{point[1]}"],
55                 frame_data[f"y_lm{point[1]}"],
56                 frame_data[f"z_lm{point[1]}"],
57             ]
58         )
59         P3 = np.array(
60             [
61                 frame_data[f"x_lm{point[2]}"],
62                 frame_data[f"y_lm{point[2]}"],
63                 frame_data[f"z_lm{point[2]}"],
64             ]
65         )
```

```

66
67     if all(
68         arr.size != 0 for arr in [P1, P2, P3]
69     ): # evitar errores si faltan puntos
70         angle_df.iat[frame, i + 1] = angle_between(P1, P2, P3)
71     else:
72         angle_df.iat[frame, i + 1] = np.nan # por si faltan landmarks
73
74 video_data_angles = pd.merge(video_data, angle_df, on="frame", how="left")
75
76 return video_data_angles

```

Función para obtener un ángulo dados tres puntos:

Python

```

1 def angle_between(P1, P2, P3): # sacar ángulo con el arccoseno
2     v1 = P2 - P1
3     v2 = P3 - P2
4
5     v1_u = unit_vector(v1).flatten() # aplsatamos para que se pueda hacer dot product
6     v2_u = unit_vector(v2).flatten()
7     return np.arccos(np.clip(np.dot(v1_u, v2_u), -1.0, 1.0)) # output en rad

```

Función para obtener el vector unitario:

Python

```

1 def unit_vector(vector): # calcular vector unitario
2     return vector / np.linalg.norm(vector)

```

D.4 Funciones relacionadas con el tratamiento de EMG

Función para importar datos de EMG:

Python

```

1 def emg_import(file_path, NUM_CHANNELS=5):
2     data_emg = np.fromfile(file_path, dtype=np.float32)
3     # data vectorized
4     data_emg = data_emg.reshape((NUM_CHANNELS, data_emg.size // NUM_CHANNELS))
5
6     return data_emg

```

Función para truncar datos de EMG según tamaño de datos de vídeo:

Python

```

1 def trimm_emg(emg_data, video_data, FPS=32, FS=1024):
2     # check if trimming required and propose automatic trimming

```

```

3     expected_ratio = FS / FPS
4     actual_ratio = emg_data.shape[1] / video_data["frame"].nunique()
5     if np.isclose(actual_ratio, expected_ratio) and actual_ratio.is_integer():
6         print("INFO: The datasets are aligned with a natural number relation.")
7         emg_data_trimmed = emg_data
8     else:
9         print(
10            """WARNING: The datasets are misaligned or have fractional relation.
11            Manual trimming is recommended, but some will be done
12            by deleting the overflow at end of EMG data."""
13        )
14     duration = video_data["frame"].nunique() / FPS
15     expected_emg_samples = int(duration * FS)
16     emg_data_trimmed = emg_data[:, :expected_emg_samples]
17
return emg_data_trimmed

```

Función para redimensionar los datos de EMG:

Python

```

1 def reshape_emg(emg_data, video_data, FPS=32, FS=1024, NUM_CHANNELS=5):
2     num_frames = video_data["frame"].nunique()
3     samples_per_frame = int(FS / FPS)
4
5     columns = [f"emg_{i}_ch_{j}" for j in range(1, 6) for i in range(32)]
6     data = np.zeros((num_frames, samples_per_frame * NUM_CHANNELS))
7     for k in range(num_frames):
8         for j in range(NUM_CHANNELS):
9             start = k * samples_per_frame
10            end = start + samples_per_frame
11            data[k, j * samples_per_frame : (j + 1) * samples_per_frame] = emg_data[
12                j, start:end
13            ]
14
15     emg_data_reshaped = pd.DataFrame(data, columns=columns)
16
return emg_data_reshaped

```

D.5 Funciones relacionadas con la fusión del dataset

Función general para la fusión del dataset: (Esta función hace llamada a todas las funciones de tratamiento anteriormente citadas)

Python

```

1 def combine_video_emg_csv(emg_path, csv_path):
2     # import and pivot video data
3     video_data = pd.read_csv(csv_path)
4     video_data = pivot_csv_table(video_data)
5     video_data = smooth_video_coord(video_data)
6     video_data = get_hand_angles(video_data)

```

```
7      # import and process emg data
8      emg_data = emg_import(emg_path)
9      emg_data = trimm_emg(emg_data, video_data)
10     emg_data = reshape_emg(emg_data, video_data)
11
12     # create dataframe with all data
13     X = pd.concat(
14         [
15             video_data.reset_index(drop=True),
16             emg_data,
17         ],
18         axis=1,
19     )
20
21
22     FILE_NAME = Path(csv_path).stem
23     csv_combined_path = "./MERGE_Export" + FILE_NAME + "_COMBINED.csv"
24     X.to_csv(csv_combined_path, index=False)
25     return
```
