

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE



**“DESARROLLO DE UN SISTEMA DE SENSORES
PARA PRÓTESIS BIÓNICAS”**

BASTIÁN EDUARDO RIVAS CORDERO

**MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO
CIVIL ELECTRÓNICO**

PROFESOR GUÍA: DR. ALEJANDRO WEINSTEIN.

PROFESOR CORREFERENTE: DR. MARCELO PÉREZ.

MAYO-2025

Agradecimientos

De partida quiero agradecer a mi familia, sobre todo a mis padres y hermano por siempre haber estado ahí y por el tremendo apoyo que me han dado a lo largo de estos años.

También quiero agradecerle a mis profesores, ayudantes y a la gente que conocí en la universidad por siempre haberme dado apoyo e inspiración para llegar hasta este punto.

Agradezco también a los chicos de Innovación y Robótica por todas las experiencias que hemos tenido juntos, las conversaciones en los bloques libres, los proyectos que hemos hecho y las cantidades industriales de café que me han dado. En particular quiero también agradecerle a la gente de LIMB, porque fue por ellos que nació esta idea de memoria.

Resumen

En este proyecto se aborda el desempeño de sensores mioeléctricos (EMG) SEN0240 fabricados por DFRobot y OyMotion, haciendo un análisis de señales de varios gestos en múltiples sesiones. Para ello, se propone un abanico de gestos, un diseño de brazalete de tres electrodos y se genera una serie de programas para los cálculos relevantes. Los datos EMG se registran a partir de tres canales, seguidos de un proceso de filtrado y normalización de las señales para extraer características relevantes. Además, se estima su espectro usando el método de Welch para analizar la frecuencia de las señales y se obtienen parámetros como la relación señal-ruido (SNR) y el valor cuadrático medio (RMS) captados por los sensores.

Para permitir el manejo de datos y facilitar el crecimiento del proyecto, se diseñó un sistema de almacenamiento de datos utilizando una base de datos SQLite, permitiendo la organización y registro de datos en diferentes sesiones y gestos. A partir de los datos almacenados, se realizaron cálculos estadísticos como la desviación estándar de la relación señal-ruido (SNR) y valor cuadrático medio (RMS) para evaluar la consistencia y precisión de los sensores. Los resultados obtenidos demuestran que los gestos de muñeca, apertura y cierre de mano son los mejor captados por los sensores. Asimismo, la utilización de herramientas como Python y SQL facilita el procesamiento y análisis de grandes volúmenes de datos, permitiendo un seguimiento detallado de las características de las señales a lo largo de múltiples sesiones.

Palabras clave: Señales EMG, Método de Welch, Relación Señal-Ruido, Procesamiento de Señales, Interfaz Hombre-Máquina, Control de Prótesis, SQLite, Python.

Abstract

In this project, the performance of myoelectric sensors (EMG) SEN0240 fabricated by DFRobot and OyMotion is addressed by analyzing signals from various gestures across multiple sessions. For this purpose, a range of gestures is proposed, a three-electrode armband design is developed, and a series of programs for relevant calculations is generated. The EMG data is recorded from three channels, followed by a process of filtering and normalizing the signals to extract relevant features. Additionally, the Welch method is used to estimate the signal spectrums, analyze the signal frequencies and obtain parameters such as the signal-to-noise ratio (SNR) and root mean square (RMS) captured by the sensors.

To allow easier data management and support the project's growth, a data storage system was designed using an SQLite database, allowing for the organization and registration of data across different sessions and gestures. Based on the stored data, statistical calculations such as the standard deviation of the SNR and RMS were performed to evaluate the consistency and accuracy of the sensors. The results indicate that wrist gestures, hand opening, and closing are best captured by the sensors. Furthermore, the use of tools like Python and SQL simplifies the processing and analysis of large volumes of data, enabling detailed tracking of signal characteristics across multiple sessions.

Keywords: EMG signals, Welch method, Signal-to-Noise Ratio, Signal Processing, Human-Machine Interface, Prosthesis Control, SQLite, Python.

Glosario

- **Base de datos:** Sistema organizado para almacenar, gestionar y recuperar datos de manera eficiente.
- **Contracción Máxima Voluntaria (CVM):** La máxima fuerza que un músculo puede generar mediante una contracción voluntaria.
- **Electrodo:** Dispositivo que permite la medición de señales eléctricas en el cuerpo.
- **Electromiografía (EMG):** Técnica para registrar y analizar la actividad eléctrica producida por los músculos.
- **Electromiografía superficial (sEMG):** Tipo de EMG que mide la actividad muscular a través de electrodos colocados en la superficie de la piel.
- **Filtro:** Dispositivo o proceso que elimina componentes no deseados de una señal.
- **Grados de libertad (DOF):** Número de movimientos independientes que puede realizar un sistema mecánico.
- **Matplotlib:** Biblioteca de Python utilizada para crear gráficos y visualizaciones de datos.
- **Microcontrolador:** Circuito integrado que incluye un procesador, memoria y periféricos de entrada/salida.
- **Normalización:** Proceso de ajustar los valores de datos para que estén dentro de un rango común.
- **Procesamiento digital de señales:** Manipulación y análisis de señales digitales para extraer información útil.

-
- **Onset:** El inicio de algo
 - **Prótesis:** Dispositivo que reemplaza una parte del cuerpo perdida o dañada.
 - **Python:** Lenguaje de programación de alto nivel utilizado para desarrollo de software y análisis de datos.
 - **Relación señal-ruido (SNR):** Comparación entre una señal deseada y el ruido de fondo.
 - **SciPy:** Biblioteca de Python para matemáticas, ciencia e ingeniería.
 - **SQLite:** Motor de base de datos SQL ligero y autónomo.
 - **Transformada Rápida de Fourier (FFT):** Algoritmo para calcular la transformada de Fourier de una señal, utilizada para analizar sus componentes de frecuencia.
 - **Valor cuadrático medio (RMS):** Valor eficaz de una magnitud eléctrica, permite conocer la potencia media. Viene definido por la ecuación 0.1

$$RMS = \sqrt{\frac{1}{N} \sum_{i=1}^N a_i^2}, \quad (0.1)$$

donde a_i representa al valor i dentro de la muestra total.

- **Valores separados por coma (CSV):** Formato de archivo para almacenar datos tabulares en texto plano, con valores separados por comas.
- **Método de Welch:** Técnica de estimación de la densidad espectral de potencia de una señal

Índice general

1.. <i>Introducción</i>	1
1.1. Objetivo general	1
1.2. Objetivos específicos	1
1.3. Estructura del documento	2
2.. <i>Marco teórico</i>	3
2.1. Estado del arte	4
2.1.1. Revisión de productos disponibles	6
2.1.2. Sensores no invasivos	7
3.. <i>Sensores EMG</i>	11
3.1. Principio de funcionamiento	11
3.2. Alternativas comercialmente disponibles	12
3.3. Alternativa escogida	15
4.. <i>Metodología</i>	18
4.1. Participantes	21
5.. <i>Validación de la fidelidad de mediciones del microcontrolador</i>	22
5.1. Entorno de validación	22
5.2. Mediciones obtenidas	23
6.. <i>Hardware usado</i>	27
6.1. Sensores Gravity Analog EMG de OyMotion	27

6.2. Brazaletes con sensores	27
6.3. Circuito de adquisición	28
7.. <i>Características de interés de las señales</i>	30
7.1. Estimación del espectro	30
7.2. Cálculo de RMS y SNR	30
8.. <i>Código y scripts usados</i>	32
8.1. Código del microcontrolador	32
8.2. Código para el procesamiento de las señales	33
8.2.1. Base de datos	33
8.2.2. Scripts generados	34
9.. <i>Preparación experimental</i>	36
10.. <i>Resultados</i>	38
10.1. Mediciones brutas por gesto	38
10.2. Normalización de señales	38
10.3. Cálculo de RMS y SNR	38
10.4. Estimación del espectro usando Welch	45
11.. <i>Conclusiones y trabajo futuro</i>	48
Anexo	58
A.. <i>Dimensiones de los sensores</i>	59
B.. <i>Código en Arduino para retornar las lecturas análogas</i>	62
C.. <i>Scripts en Python</i>	64
C.1. Lectura de 3 canales y onset	64
C.2. Consultar los gestos registrados en la base de datos	69

C.3. Graficar los 3 canales de un gesto en bruto	71
C.4. Normalizar los gestos	76
C.5. Obtener el espectro de gestos registrados en bruto	97
C.6. Generar la tabla de FFT	102
C.7. Calcular la desviación estándar	114

Índice de figuras

2.1. Uno de los Hero Arm desarrollados por OpenBionics, [1]	3
2.2. La 'Eiserne Hand' o Mano de hierro del caballero Götz von Berlichingen. Imagen tomada de Wikimedia Commons	4
2.3. Banda WyoFlex impresa en 3D [2](a) Vista lateral de la WyoFlex en el antebrazo, (b) Vista superior de la WyoFlex	8
2.4. Diadema MindWave desarrollado por NeuroSky [3]	9
3.1. Diagrama de bloques con las partes más importantes de un sensor EMG. Dependiendo del caso puede ser deseable usar la señal tal cual o su envolvente.	12
3.2. Preparación de sensor Myoware sin autoadhesivos. Tomado de la página oficial de Myoware	17
3.3. Kit del Gravity Analog EMG Sensor	17
4.1. Imágenes de gestos hechos	20
4.2. Diagrama con la configuración a usar en las mediciones	21
5.1. Circuito para la validación de uno de los canales del ADC	23
5.2. Señal original medida desde osciloscopio	24
5.3. Medición obtenida con un Arduino Nano de una señal con frecuencia de muestreo de 500 Hz	24
5.4. Medición obtenida con un Arduino Nano de una señal con frecuencia de muestreo de 1 kHz	25

5.5. Medición obtenida con una ESP32 de una señal con frecuencia de muestreo de 500 Hz	25
5.6. Medición obtenida con una ESP32 de una señal con frecuencia de muestreo de 1 kHz	26
6.1. Modelos 3D de las partes del brazalete	28
6.2. Circuito de adquisición de señal y alimentación de sensores	29
9.1. Ubicación de los sensores en el antebrazo	37
10.1. Primer set de capturas brutas hechas. Las líneas verticales representan al inicio y fin de la ejecución del gesto	39
10.2. Segundo set de capturas brutas. Las líneas verticales representan al inicio y fin de la ejecución del gesto	40
10.3. Normalización del gesto "Puño" usando la CVM y un filtro de segundo orden a 150 Hz	41
10.4. Primer set de espectros calculados con las señales sin filtrar	46
10.5. Segundo set de espectros calculados con las señales sin filtrar	47
A.1. Dimensiones de las placas de electrodos	60
A.2. Dimensiones de las placas de amplificadores	61

Índice de cuadros

2.1. Ejemplos de algunas prótesis desarrolladas	6
2.2. Resumen de los tipos de sensor investigados	10
3.1. Alternativas de sensores investigadas	14
3.2. Resumen de ventajas y desventajas de cada tipo de electrodo	16
3.3. Comparación de distintos productos	16
3.4. Comparación entre los materiales necesarios con la Myoware Muscle Sensor y Gravity Analog EMG Sensor para usar electrodos secos	17
5.1. Características de interés de las placas Arduino Nano y ESP32	22
8.1. Categorías dentro de la tabla raw	33
8.2. Categorías dentro de la tabla norm	34
8.3. Categorías dentro de la tabla fft	34
10.1. Amplitud por gesto	42
10.2. Valores promedio de RMS	43
10.3. Valores promedio de SNR	44
10.4. Desviación estándar de SNR por canal	44

1. INTRODUCCIÓN

En este trabajo se hace un estudio de las distintas maneras en que se pueden tomar mediciones desde el cuerpo humano para permitir el control de prótesis, así como su disponibilidad, factibilidad y desempeño en la captura de datos. En particular, se trabaja con sensores de electromiografía superficial (sEMG) SEN0240 que usan electrodos secos y se analiza a detalle su funcionamiento al ejecutar una abanico de gestos comúnmente usados. Usando una placa microcontroladora comercialmente disponible, se capturan dichos datos y se procesan digitalmente para analizar las características de los sensores y qué gestos reconocen mejor.

1.1. Objetivo general

Caracterizar sensores de electromiografía SEN0240 de DFRobot y OyMotion en base a sus salidas frente a un conjunto de gestos de mano.

1.2. Objetivos específicos

Se plantean los siguientes objetivos específicos:

- Diseñar un dispositivo con forma de brazalete compuesto por sensores mioeléctricos SEN0240 montable en un antebrazo.
- Tomar repetidas muestras en un antebrazo con distintos gestos.
- Comparar la salida de los sensores para distintos gestos viendo niveles de ruido, amplitud y SNR.

-
- Hacer un análisis estadístico con media y desviación estándar para cada gesto usando gráficos.

1.3. Estructura del documento

En el capítulo 2 se entrega el contexto y el marco teórico que envuelve a este trabajo, el estado del arte en cuanto a prótesis y sensores, como también las alternativas vistas para permitir el control de prótesis. El capítulo 4 describe la metodología general que se usa en el trabajo, los gestos a usar y un flujo general de trabajo de las señales.

El capítulo 6 describe todas las componentes físicas usadas, mostrando modelos, diagramas y conexiones hechas ejecutar y replicar el trabajo hecho. Además, en el capítulo 5 se presentan opciones de microcontroladores usables, sus características y los criterios usados para escoger uno de ellos.

El capítulo 7 expone las características de interés de las señales y los factores de interés para calcularlas.

En el capítulo 8 se explica el funcionamiento del software y los programas generados durante el trabajo, tanto para el microcontrolador como para el postprocesado y visualización de las señales.

El capítulo 9 especifica cómo se disponen los componentes del entorno de prueba luego de haberlos introducido en los capítulos anteriores, como también los cuidados que hay que tener antes de llevar a cabo cada experimento.

Finalmente, el capítulo 10 despliega los resultados más relevantes tras las sesiones de captura y procesado, y en el capítulo 11 se presentan conclusiones relevantes para los objetivos y se propone un trabajo futuro que utilice como base lo expuesto en este proyecto.

2. MARCO TEÓRICO

A lo largo de los años se han desarrollado varias formas para restituir extremidades perdidas, siendo algunos ejemplos los ganchos, prótesis mecánicas y, en un frente más enfocado en la innovación, las prótesis biónicas [4]. Las últimas, en particular, han demostrado ser inaccesibles para la mayoría de los pacientes discapacitados debido a su alto precio y, en algunos casos, la mala relación costo-beneficio en su implementación [5]. Dada esta problemática, han aparecido varias fundaciones internacionales que intentan facilitar el acceso a esta clase de dispositivos por medio de proyectos *open source* e impresión 3D, como e-NABLE [6] y OpenBionics en sus primeros años [1], desarrollando manos como el Hero Arm de la figura 2.1, que son controlados por sensores musculares. La forma de operar de estos proyectos es dejar en repositorios abiertos a la comunidad sus modelos 3D y permitir que cualquier interesado pueda descargarlos e imprimirlos por su cuenta. Además, entregan un listado de materiales para replicar alguno de los modelos disponibles en sus páginas.



Fig. 2.1: Uno de los Hero Arm desarrollados por OpenBionics, [1]

Latinoamérica tampoco se ha mostrado indiferente respecto a este tema. En la

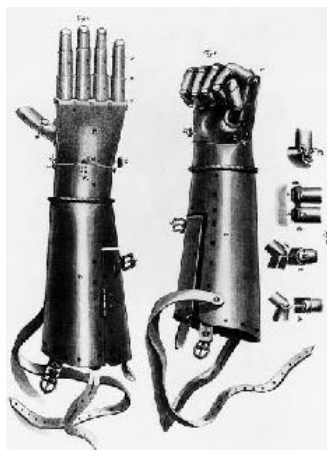


Fig. 2.2: La 'Eiserne Hand' o Mano de hierro del caballero Götz von Berlichingen. Imagen tomada de Wikimedia Commons

región se han llegando a generar organismos como la Fundación Prótesis 3D Chile, que ofrecen sus propias impresiones en 3D y cursos sobre el tratamiento que deben recibir sus beneficiarios [7].

2.1. Estado del arte

Una parte importante del desarrollo de estos equipos es cómo el paciente puede interactuar con ellos y usarlos de una manera cómoda. Algunas prótesis tradicionales se basan en principios mecánicos para ser manipuladas con movimientos de otras partes del cuerpo. Las primeras prótesis eran dispositivos puramente mecánicos hechos de hierro que usaban tiras de cuero para montarse en el cuerpo y transmitir tensión para hacer movimientos básicos, como la mano del general romano Marcus Sergius en el año 77 dC o la *Eiserne Hand* del caballero alemán Götz von Berlichingen cerca del 1505 en la figura 2.2 [8].

Las prótesis mecánicas más modernas suelen hacerse en conjunto con el paciente, teniendo en cuenta sus necesidades y el uso que se les quiera dar. En algunos casos pueden emplearse manos dedicadas a levantar peso y tener agarres firmes [9] o permitir el manejo de dispositivos táctiles. No solo se limitan a tener una forma parecida a la de una mano orgánica, sino que pueden emplearse ganchos por su simpleza y

capacidad de permitir movimientos precisos [10].

Al introducir la parte electrónica, se suelen usar motores o actuadores para mover el dispositivo con una cierta cantidad de grados de libertad (DOF, por su sigla en inglés). La transmisión de movimiento puede hacerse usando un sistema de cuerdas que imiten a los tendones humanos o poniendo motores directamente en las articulaciones, dependiendo del diseño. Además, se requiere alguna clase de señal de referencia para que la parte actuadora ejecute los movimientos que el usuario desea efectuar [11]. Los movimientos a ejecutar en una mano con un DOF son tan sencillos como la apertura y cierre de la misma, pudiendo usar la amplitud de la señal como control de cuánto debería apretar la mano. En casos de más DOF se deben emplear técnicas como el control de trayectorias [12] o gestos preprogramados [13] para que el paciente pueda controlar los otros dedos.

Las formas típicas para manejar prótesis biónicas de forma no invasiva consideran técnicas como el control mioeléctrico superficial (sEMG), potenciómetros, resistencias sensibles a la fuerza (FSR) y la electroencefalografía (EEG). También existen otras técnicas usando acelerómetros, ultrasonido o sonomiografía (SMG) [14, 15]. Dependiendo del tipo de sensor puede que se favorezca la simpleza de sensores con poca capacidad de reconocer gestos [16], o captación de señales muy susceptibles al ruido pero que dan más información para detectar gestos [17].

Actualmente, la que ha sido mayormente implementada en modelos comerciales ha sido EMG. Con esta clase de sensores se miden los potenciales eléctricos generados al tensar los músculos, dando típicamente valores en torno a los μV que deben ser acondicionados. Suele venir acompañada de técnicas de Machine learning (ML) o redes neuronales artificiales (ANN) para distinguir mejor los gestos, sobre todo en prótesis más complejas [18]. Puede verse también acompañada de unidades de medición inercial (IMU) o acelerómetros para mejorar la detección de gestos [19].

2.1.1. Revisión de productos disponibles

Los proyectos investigados se presentan ordenados en la tabla 2.1 según el tipo de sensor que utilizan. La mayoría de los investigados están comercialmente disponibles, incluyendo un ejemplo académico. Se puede notar que la técnica de sensado de preferencia es EMG para los modelos comerciales. En implementaciones más académicas se ensaya con otra clase de sensores, como EEG.

Nombre	Sensor usado	DOF	Rpta háptica	Fabricante
Zero Arm [17]	EEG	4	Sí, tacto con servos	Autores del paper
BionicoHand [20]	EMG	1	No	Bionico
AxonHook [21]	EMG	1	No	Ottobock
ETD2 [22]	EMG	1	No	Arm Dynamics
Low-Cost Robotic Arm... [19]	EMG	6	No	Autores del paper
i-Limb Ultra/Quantum [23]	EMG	7	No	Össur
Brunel Hand 2.0 [24]	EMG	9	No	OpenBionics
Dextra [25]	EMG	15	No	DIY - Open source
An EMG Controlled Bionic Arm... [18]	EMG	15	No	Paper
beBionic [26]	EMG	-	No	Ottobock
Michelangelo [27]	EMG	-	No	Ottobock
SensorHand Speed [28]	EMG	-	No	Ottobock
VINCENT young3+ [29]	EMG	-	No	Vincent Systems
Hero Arm [1]	EMG	3 ó 4	No	OpenBionics
i-Digits [30]	EMG	Variable	No	Össur
Taska Hand [31]	EMG		No	Arm Dynamics
Ability Hand [32]	EMG	6	Sí, tacto	Psyonic
TrueLimb [33]	EMG	-	Sí, tacto con piezo	Unlimited Tomorrow
DEKA/LUKE Arm [34]	IMU/ EMG/ Switches	10	No	Mobius Bionics
Gancho [35]	Sistema mecánico	1	No	Steeper
Eiserne Hand [8]	Sistema mecánico	1	No	Götz von Berlichingen
Single-DoF Prosthetic... [9]	No especifica	1	No	Autores del paper

Tab. 2.1: Ejemplos de algunas prótesis desarrolladas

2.1.2. Sensores no invasivos

El trabajo con bioseñales para controlar una prótesis implica adquirir dicha señal, adecuarla con filtraje y amplificación, extraer sus características y, finalmente, clasificarlas o hacer una regresión. El desafío de adquirir esas señales ha sido abarcado de distintas formas, usando técnicas tanto invasivas como no invasivas.

Respecto a los sensores no invasivos, los más populares en la industria son los sEMG, debido a su variedad de aplicaciones en rehabilitación y el control de dispositivos. Esta clase de dispositivos suelen venir en forma de brazaletes con varios electrodos puestos en la piel distribuidos en forma radial, algunos llegando a tener hasta 8 canales [2]. Algunas implementaciones consideran la adición de una IMU para medir la aceleración normal a la superficie del brazo para mejorar la detección de movimientos musculares [19]. La señal saliente de los sensores debe acondicionarse para ser procesada por un ADC y generar una actuación proporcional al voltaje medido, o bien, usar técnicas de Deep Learning, ANN o ML para extraer las características de la flexión de cada dedo y permitir un mayor rango de gestos [13].

Algunos ejemplos de esta clase de productos es la Myo Armband de Thalmic Labs y la WyoFlex desarrollada por Gomez-Correa M. y Cruz-Ortiz D. [2]. La primera fue descontinuada debido a que su frecuencia de muestreo no alcanzaba a captar bien las señales musculares, lo que provocaba que existiera una latencia poco natural y mala respuesta por parte de los equipos controlados, inspirando a que se desarrollen otras alternativas con mejor desempeño. También ha sido de interés desarrollar soluciones minimalistas en términos de hardware, proponiendo alternativas con menor cantidad de canales y evaluando su desempeño en reconocer gestos [36]. Por otra parte, también ha sido un desafío tener un tipo de electrodos que permita tomar mediciones de los músculos sin tener que depender de un gel conductor debido a su incomodidad y posibles riesgos a la piel del paciente, proponiendo el uso de electrodos superficiales secos para permitir el uso cotidiano de estos dispositivos [2].

Analizando el caso particular de la WyoFlex para el proceso de sensado, ha pro-

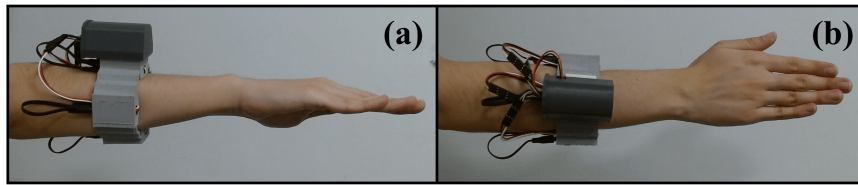


Fig. 2.3: Banda WyoFlex impresa en 3D [2](a) Vista lateral de la WyoFlex en el antebrazo, (b) Vista superior de la WyoFlex

puesto usar dos brazaletes con dos sensores cada uno, siendo controlados por una ESP32 y transmitiendo los datos inalámbricamente. La frecuencia de muestreo llega a ser de 1600 Hz para garantizar que no se pierda señal en el proceso. Finalmente, se hace una evaluación de su desempeño usando 6 gestos y reconociéndolos usando una red neuronal artificial. Los sEMG en sí son módulos Gravity Analog EMG (SEN0240) desarrollados por DFRobot y OYMotion, con salidas entre 0 y 3 V y un voltaje de referencia de 1,5V [2]. La banda desarrollada por este grupo se presenta en la figura 2.3.

Otra propuesta hecha dentro del área han sido los EEG. Estos consisten en electrodos puestos en puntos estratégicos en la cabeza del usuario para captar las señales eléctricas cerebrales producidas por la actividad neuronal en distintas bandas de frecuencia. Posee la ventaja de tener buena resolución temporal y tener un precio competitivo, pero tiene el problema de tener una señal débil y sensible a interferencias, lo que provoca que tengan un peor desempeño que otras alternativas más tradicionales [37]. Usualmente, el procesamiento de esta clase de señales se acompaña con machine learning para poder clasificar el tipo de gesto hecho y producir una señal de control. Algunas implementaciones comerciales contemplan diademas o *headsets* que se montan en la cabeza del usuario, como MindWave de NeuroSky [3] o Insight de EMOTIV [38]. Una forma típica de esta clase de instrumentos se presenta en la figura 2.4. Una forma estudiada en la academia para hacer interfaces humano-máquina es la sonomiografía (SMG), que consiste en usar sensores de ultrasonido para medir las contracciones de músculos tanto superficiales como los más profundos. Posee la ventaja de tener baja latencia y distinguir más detalles de cada músculo. A pesar de



Fig. 2.4: Diadema MindWave desarrollado por NeuroSky [3]

ser una técnica prometedora, sigue requiriendo más refinamiento porque suelen usarse sensores voluminosos y bastante consumidores de potencia [11] [15].

Otra técnica analizada en la academia es la mecanomiografía (MMG), que consiste en usar distintos tipos de transductores para captar las vibraciones y/o contracciones hechas en su actividad. Suelen usarse acelerómetros, micrófonos o sensores piezo-eléctricos de contacto para captar esta clase de señales [39]. Dada su naturaleza de capturar movimientos mecánicos, no requiere usar electrodos ni gel conductor para funcionar. Sin embargo, debido a su baja frecuencia de muestreo puede generar que los movimientos de actuación sean más lentos.

Tipo de sensor	Modo de funcionamiento	Ventajas	Desventajas
sEMG [13]	Capta señales eléctricas del movimiento de músculos	Relativamente bajo costo, común de ver en prótesis	Interferencia entre músculos, susceptible a cansancio y sudor
EEG [37]	Capta ondas cerebrales de la actividad neuronal	Buena resolución temporal	Características difíciles de discernir, requiere muchos canales
SMG [15]	Hace una imagen en ultrasonido de los músculos de la zona	Baja latencia, buena relación señal-ruido (SNR)	Alto consumo de potencia, muy grande
MMG [39]	Percibe las vibraciones de la actividad muscular	No requiere electrodos ni gel, mejor calidad que sEMG	Alta latencia, requiere mucha calibración

Tab. 2.2: Resumen de los tipos de sensor investigados

3. SENSORES EMG

Habiendo visto las distintas técnicas no invasivas de captación de señales biomédicas, se puede concluir que las más comercialmente disponibles son la electromiografía superficial y, en menor medida, los electroencefalogramas. El resto de las técnicas, pese a ser novedosas, siguen teniendo ciertas limitaciones y faltas de perfeccionamiento que las retienen dentro del mundo académico, por lo que en este estudio se enfatiza en la electromiografía.

3.1. *Principio de funcionamiento*

En términos generales, la electromiografía consiste en la medición, registro y análisis de señales mioeléctricas. Estas señales se producen ante las variaciones fisiológicas del estado de las membranas de las fibras musculares, llegando a ser del orden de las decenas de milivoltios y con frecuencias entre 10 y 500 Hz [40]. En el caso particular de la electromiografía superficial o sEMG, las señales pueden captarse desde la piel usando electrodos secos o húmedos con la ayuda de un gel conductor para mejorar la calidad de las mismas.

Respecto al posicionamiento de los electrodos, comúnmente se usan 3 para usar una configuración bipolar. Esta configuración consiste en tener un primer electrodo al centro del músculo y un segundo electrodo a 1-3 cm de distancia del primero. Se tiene también un tercer electrodo en una zona donde no haya musculatura para ser usada como referencia al ser eléctricamente neutra. De este método se capta la diferencia entre ambas mediciones, dando la ventaja de eliminar ruido entre ambos por el CMRR del amplificador operacional [41].

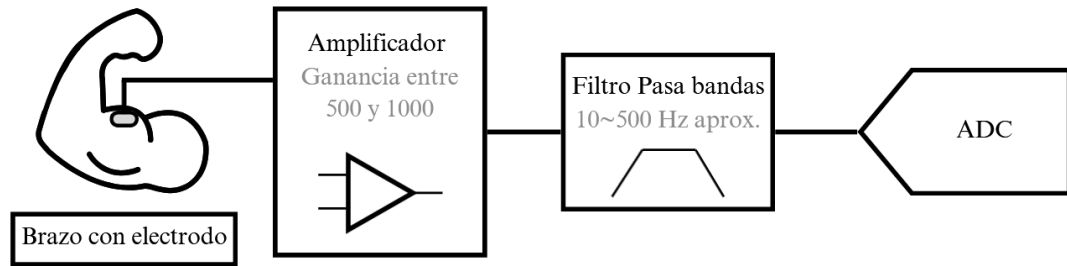


Fig. 3.1: Diagrama de bloques con las partes más importantes de un sensor EMG. Dependiendo del caso puede ser deseable usar la señal tal cual o su envolvente.

Luego de captar los voltajes generados por la musculatura, se requiere adecuar las señales para que puedan ser captadas por otros dispositivos y ser procesadas para su posterior caracterización. Para ello, se suele amplificar la señal con una ganancia típicamente entre 500 y 1000, y se pasa por un filtro pasa bandas para captar las partes de interés de la señal [40]. Según la aplicación y posterior tratamiento de la señal, puede que convenga agregar un bloque adicional para captar la envolvente de la señal. El proceso de captación se resume en la figura 3.1, donde se muestran las partes básicas para 1 canal de señales.

Tras digitalizar las señales musculares, el procesamiento de las mismas puede hacerse con una multitud de estrategias posibles. En análisis sencillos se suele ver la amplitud de la envolvente de las señales [42] o, en sistemas más sofisticados, aplicar algoritmos de Machine Learning y caracterizar los gestos que esté haciendo el usuario [43]. Algunos algoritmos usados son Support Vector Machine (SVM), Linear Discriminant Analysis (LDA) y Multi-Layer Perception (MLP) [44], usando un computador con un programa como MATLAB para ejecutarlos.

3.2. Alternativas comercialmente disponibles

Dado el análisis y la manera de operar de esta técnica de medición, se investigaron algunos sensores mencionados en los papers revisados para el estado del arte y en los modelos de prótesis más populares. De lo anterior, se vieron tanto unidades de

sensores individuales como algunas implementaciones en forma de brazaletes usables en la parte del brazo o antebrazo. Como criterio adicional de filtrado de opciones, se descartaron modelos que ya no se fabrican o que, de ser diseños nuevos, no tengan un repositorio con sus archivos disponibles, por lo que se eliminó la Myo Armband de Thalmic Labs a pesar de su popularidad en ámbitos de investigación, y se descartó la WyoFlex por no ser directamente replicable, pero sí puede servir como fuente de inspiración porque usa un par de sensores Gravity Analog EMG en su implementación.

De las opciones restantes quedan 3 unidades de sensor y 2 brazaletes. El análisis de estos elementos contempla su precio, disponibilidad, la necesidad de gel conductor para mejorar la señal, su tamaño, la cantidad de canales para los brazaletes, y la salida retornada. Adicionalmente, se notó que los brazaletes complementaron su salida con las mediciones de IMUs, resumiendo dichas características en la tabla 3.1.

Las unidades de sensores Myoware y Muscle Sensor V3 consisten en placas con un amplificador de instrumentación, un filtro incorporado y conectores para cables de electrodos. Están pensadas para ser conectadas directamente al ADC de un microcontrolador porque retornan una salida análoga dentro de los rangos de voltaje típicos de operación para esta clase de unidades. Los electrodos típicamente usados con estas unidades son desechables. Por otra parte, la implementación propuesta para el OYmotion contempla 2 tipos de placa. El primer tipo incorpora electrodos secos, visibles como rectángulos metálicos que van en contacto directo con la piel, mientras que el segundo maneja la amplificación, filtrado y transmisión de la señal.

En cuanto a los brazaletes se encontró el Mindrove armband, que posee 8 canales de EMG y 2 adicionales de una IMU. Además, ofrece un SDK que permite trabajar con todas las señales recibidas. Otro brazalete encontrado fue uno desarrollado en un paper [45] como respuesta al alto precio que pueden llegar a alcanzar esta clase de equipos. Posee 10 canales, una IMU de 9 ejes y circuitos integrados hechos a la medida para manejar la adquisición de datos. Ofrece todos los archivos necesarios en un repositorio abierto al público, por lo que el costo se concentra principalmente en los materiales usados y en la fabricación del equipo. En cuanto a los materiales,

se pudo ver que buena parte del precio está concentrado en los amplificadores de instrumentación necesarios.

Nombre	Myoware Muscle Sensor [41]	Gravity Analog EMG Sensor [46]	Muscle Sensor V3 (basado en AD8221) [47]	Mindrove armband 8 channel [48]	3DC Arm-band [45]
Tipo de producto	Unidad de sensor	Unidad de sensor	Unidad de sensor	Brazaletes con sensores	Brazaletes con sensores
Precio (USD)	39.95	49.50	31.79	729.00	150
Disponibilidad	Compra internacional	Compra internacional	Compra nacional	Compra internacional, 6-8 semanas desde encargo	DIY. Sujeto a disponibilidad de componentes + impresión de PCB
¿Requiere gel conductor?	Sí	No	Sí	No	No
Tamaño	38 x 36 mm	22 x 35 mm	25,4 x 25,4 mm	Ajustable	Ajustable
N° Ch	1	1	1	8+2 IMU	10 + 1 IMU
Forma de onda de salida	EMG amplificado, rectificado y/o envolvente	EMG amplificado	EMG amplificado, rectificado y suavizado	EMG amplificado x8 + IMU x2	EMG amplificado x10 + IMU

Tab. 3.1: Alternativas de sensores investigadas

Respecto a los tipos de electrodos a usar se han visto dos alternativas posibles. Por un lado, se presentan los electrodos desechables, que se adhieren a la piel con autoadhesivos y se puede mejorar su impedancia usando gel conductor. Tienen la ventaja de ser bastante comunes y retornar una señal clara, pero su uso prolongado puede traer problemas a la piel e incomodidad al tener pegamento y gel. Además, existe la degradación de la señal al cabo de un tiempo por el secado del gel [49].

Por otra parte, para eliminar las incomodidades que pueden generar los electrodos autoadhesivos, se han visto alternativas como electrodos secos y textiles. Los electrodos secos consisten en cuadrados metálicos en contacto directo con la piel, cuya señal puede pasar por un preamplificador para mejorar su desempeño [50]. Sin embargo, dada su construcción, la señal captada puede sufrir de mayor ruido al tener peor conducción con la piel de la que tendrían de haber presente algún gel o pegamento, por lo que es importante que estén con suficiente presión para capturar señal pero sin incomodar al usuario. De la investigación hecha se vieron como una solución común para el armado de brazaletes mioeléctricos, como la Myo Armband, 3DC Armband y en unidades como el Gravity Analog EMG Sensor. Finalmente, los electrodos textiles se proponen como una alternativa más gentil con la piel del usuario al no tenerla expuesta a sustancias viscosas por mucho tiempo [51]. Tienen la ventaja de ser reutilizables y de ser relativamente fáciles de fabricar al requerir una manga elástica y una tela conductora, pero no son tan resistentes ni tan precisos como las otras alternativas. A modo de resumen, se presenta la tabla 3.2 con las ventajas y desventajas de cada caso.

3.3. *Alternativa escogida*

Para trabajar los distintos sensores EMG investigados hasta el momento, se busca aquel que sea más conveniente para tomar mediciones musculares y que retorne una salida que permita una posterior caracterización del mismo. Para ello, la elección se hace en cuanto a un precio factible (menos que 300 USD), la facilidad en la que se pueden obtener y la salida que retornan en caso de tener filtros o detectores de

Tipo de electrodo	Ventajas	Desventajas
Desechables	■ Buena calidad de señal ■ Permite el uso de gel conductor ■ Fáciles de encontrar	■ Posibles daños a la piel por uso prolongado ■ Requieren reemplazos periódicos
Seco metálico	■ Reutilizables ■ Mediciones consistentes a pesar de mucho tiempo de uso	■ Deben estar a presión para tener mejor mediciones ■ Menor calidad de señal
Seco textil	■ Reutilizables ■ Poco daño a la piel	■ Peor calidad de señal ■ Poco resistentes a impactos físicos

Tab. 3.2: Resumen de ventajas y desventajas de cada tipo de electrodo

envolvente, ponderando una cierta cantidad de puntaje en cuanto a qué tan bien cumple con cierta característica. También se considera como puntos a favor la existencia de herramientas o alguna librería que facilite el trabajo con el sensor, armándose así la tabla 3.3.

Nombre	Precio	Disponibilidad	Forma de onda de salida	Herramientas adicionales
Myoware Muscle Sensor [41]	✓	✓✓	✓✓	✓✓
Gravity Analog EMG Sensor [46]	✓✓	✓✓	✓✓	✓
Muscle Sensor V3 (basado en AD8221) [47]	✓✓	✓	✓✓	x
Mindrove armband 8 channel [48]	x	✓	✓✓	✓✓
3DC Armband [45]	✓✓	x	✓✓	x

Tab. 3.3: Comparación de distintos productos

De los resultados de la tabla 3.3 se puede obtener que las alternativas de Myoware y Gravity Analog EMG son las óptimas, por lo que se hace otra comparación entre



(a) Placa principal del sensor (b) Electrodo seco sugerido Myoware

Fig. 3.2: Preparación de sensor Myoware sin autoadhesivos. Tomado de la página oficial de Myoware



Fig. 3.3: Kit del Gravity Analog EMG Sensor

ambas para analizar cuál es la más conveniente a la hora de implementar electrodos secos, obteniéndose así la tabla 3.4. A pesar de que la unidad Myoware sea más barata, la inclusión de electrodos secos, banda elástica y cables hace que la opción de Gravity sea más atractiva, como se presenta en la figura 3.3. Por otra parte, para usar electrodos secos con Myoware se propone el armado que se presenta en la figura 3.2 presentado en su sitio oficial, lo que agrega costos extra a esa opción.

Nombre	Precio total (USD)	Electrodos secos
Myoware Muscle Sensor + Myoware Cable Shield + tejido conductivo	39.95 + 5.95 + 22.41	Fabricar con materiales
Gravity: Analog EMG Sensor	49.50	Incluidos

Tab. 3.4: Comparación entre los materiales necesarios con la Myoware Muscle Sensor y Gravity Analog EMG Sensor para usar electrodos secos

4. METODOLOGÍA

En las mediciones de este estudio se considera el contexto de amputaciones del tipo transradial, es decir, a nivel del antebrazo. Se trabaja bajo el supuesto de que el paciente ya recibió una terapia y que puede tensar los músculos de esa zona. Para el manejo de un antebrazo protésico, se define un conjunto de gestos que permitan al usuario hacer acciones cotidianas, como estirar dedos, hacer un puño, movimientos de muñeca y un agarre de pinza, basados en [18]. Se agregan, además, gestos en los que los músculos de interés se tensan a tope para tener puntos de comparación de cada canal.

En este trabajo se usan tres sensores, cada uno de ellos asociados a músculos o grupos musculares que controlan gestos comúnmente efectuados. Para montarlos se confecciona un brazalete que mantenga los sensores a presión en el brazo y pueda mantener las placas de los sensores en una posición que sea cómoda para el usuario. Considerando que se apunta a poder manejar una prótesis y que esta debe otorgarle comodidad al usuario, se prefiere usar electrodos secos.

La toma de muestras consiste en ejecutar una secuencia de distintos gestos para ir registrándolos en el disco duro de un computador para su posterior análisis. Los gestos hechos en cada caso se hacen con el brazo apoyado en una superficie y en periodos cortos de tiempo para evitar los efectos de la fatiga muscular. Para facilitar el posterior trabajo con las señales en Python y brindar mayor escalabilidad, se registran los datos capturados en una base de datos SQLite [52] donde se anotan distintas entradas para cada canal usado, el nombre gesto es el que se está ejecutando en el momento, la frecuencia de muestreo utilizada, la fecha de captura y un número identificador único para cada gesto. La elección de SQLite en lugar de otros formatos como valores sepa-

rados por comas (CSV, por sus siglas en inglés) se hace por la flexibilidad que ofrece a la hora de etiquetar datos y las herramientas que ofrece.

El análisis de las señales considera calcular su voltaje efectivo (RMS por sus siglas en inglés), relación señal-ruido (SNR) y extraer las características de cada gesto en el dominio del tiempo, como la amplitud de la señal y el nivel de ruido presente. Se toman los parámetros propuestos en [53], principalmente la amplitud de la señal y amplitud del ruido. Adicionalmente, se aplica el análisis propuesto en [42] para validar el trabajo hecho usando un análisis más cercano a un caso de uso.

La adquisición de señales se hace usando una placa microcontroladora con varias entradas análogas. Luego, se almacenan los datos en una base de datos con las entradas anteriormente mencionadas. Finalmente, se obtienen valores promedio, desviación estándar, se estima su espectro y se obtienen gráficos para analizar cómo es el comportamiento de los sensores ante los movimientos ejecutados.

Se usa un total de 18 gestos inspirados en la metodología de [54], agregando las categorías de reposo y las contracciones máximas voluntarias correspondientes a cada canal. Para capturar las señales de reposo se deja el brazo apoyado en una mesa con el brazalete puesto, pero con la muñeca posicionada sobre un objeto para evitar que la zona cercana al brazalete toque la superficie para minimizar interferencias. Las referencias de los movimientos de gestos se presentan en la figura 4.1.

Se incluyen las contracciones voluntarias máximas (CVM) asociadas a los músculos de cada canal como se sugiere en [42] para ser usadas en el procesamiento de las señales. Dentro del documento se ofrecen códigos con los que se pueden filtrar y normalizar las señales según su CVM para ser usadas en comparaciones.

Un diagrama de bloque con el flujo de datos a grandes rasgos se presenta en la figura 4.2, donde se muestra el microcontrolador, parte del procesado en Python y los posteriores registros en la base de datos. La entrada del ADC se conecta directamente al amplificador del sensor. El filtrado y postprocesado se hace usando un filtro buterworth IIR de segundo orden de la biblioteca SciPy con una frecuencia de corte de 150 Hz para tener buena eliminación de ruido de alta frecuencia sin tener demasiada



Fig. 4.1: Imágenes de gestos hechos

complejidad en los cálculos. La cantidad de muestras varía según la duración de cada gesto, siendo de un valor que ronda las 1000 muestras dado que se usa una frecuencia de muestreo de 1 kHz y la ejecución de los gestos dura alrededor de 1 segundo.

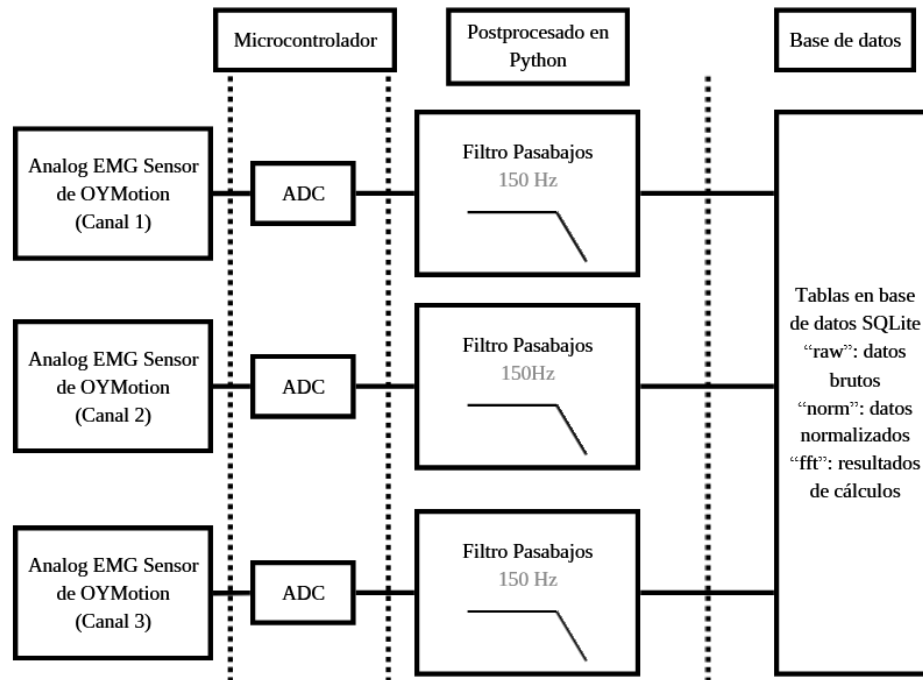


Fig. 4.2: Diagrama con la configuración a usar en las mediciones

4.1. Participantes

Las muestras se obtienen del brazo izquierdo un sujeto masculino de 24 años, diestro y sin antecedentes de discapacidad.

5. VALIDACIÓN DE LA FIDELIDAD DE MEDICIONES DEL MICROCONTROLADOR

Para caracterizar los sensores de forma que no existan factores externos que empeoren las señales, se deben verificar las opciones de placas que puedan ser usadas para adquirir los datos. Para verificar la calidad de los ADCs usados en la captura de las señales, se comparan los ADCs de dos microcontroladores. Dada su disponibilidad, se compara la señal recibida usando un Arduino Nano y la de un ESP32 Devkit V1 WROOM32. Las características de interés con las que comparar son la frecuencia de reloj, la resolución de sus ADCs, el rango de voltajes de entrada y el número de canales. La información para ambas placas se resume en la tabla 5.1.

Característica	Arduino Nano	ESP32 Devkit V1
Frecuencia de reloj	16 MHz	80-240 MHz
Resolución de ADC	10 bits	12 bits
Rango de entrada	0-5 V	0-3.3 V
Canales analógicos	8	15

Tab. 5.1: Características de interés de las placas Arduino Nano y ESP32

5.1. Entorno de validación

Considerando que se usan 3 canales para las mediciones de EMG, se hacen pruebas con un osciloscopio de 4 canales y un generador de señales para obtener formas de onda similares a las que se espera recibir desde el sensor muscular. Las señales de prueba a usar son sinusoidales que rondan entre los 50 y 150 Hz, con un offset de $1,7V_{DC}$ y voltaje peak-to-peak de $0,3V_{pp}$.

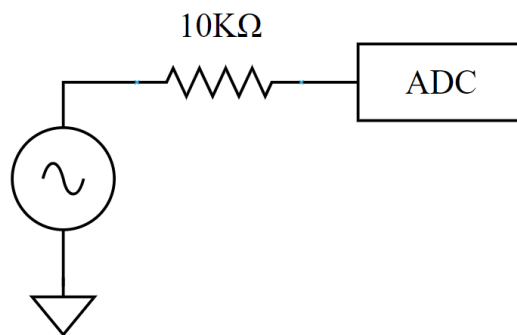


Fig. 5.1: Circuito para la validación de uno de los canales del ADC

La verificación de los microcontroladores se hace tomando mediciones por monitor serial y graficando lo recibido. Simultáneamente, se miden las entradas a los ADC de la placa con el osciloscopio y se comparan las formas de onda de ambos casos, usando al osciloscopio como referencia. Para tomar las mediciones se usa el circuito de la figura 5.1 para cada canal, donde la fuente alterna representa al generador de señales, y se incluye una resistencia para evitar dañar al ADC con una sobrecarga de corriente.

5.2. Mediciones obtenidas

La señal original de 150 kHz que se envió a las placas se presenta en la figura 5.2. Las mediciones variando las placas y las frecuencias de muestreo se presentan en las figuras 5.3 a 5.6.

De las mediciones se obtiene que Arduino Nano presenta los registros más consistentes, viéndose la mayoría de las distorsiones asociadas a las frecuencias de muestreo. A pesar de que en la figura 5.3 la señal presenta una distorsión notoria al comparar con la referencia, esta no presenta las mismas deformaciones que presenta la ESP32 en la figura 5.6, por lo que se decide continuar con Arduino. Un factor que se debe tener en consideración al replicar estos resultados es que existen diferencias de offset entre las placas utilizadas. En el caso de la ESP32 se obtiene un offset de 1,65 V, mientras que Arduino registra uno de 1,9 V.

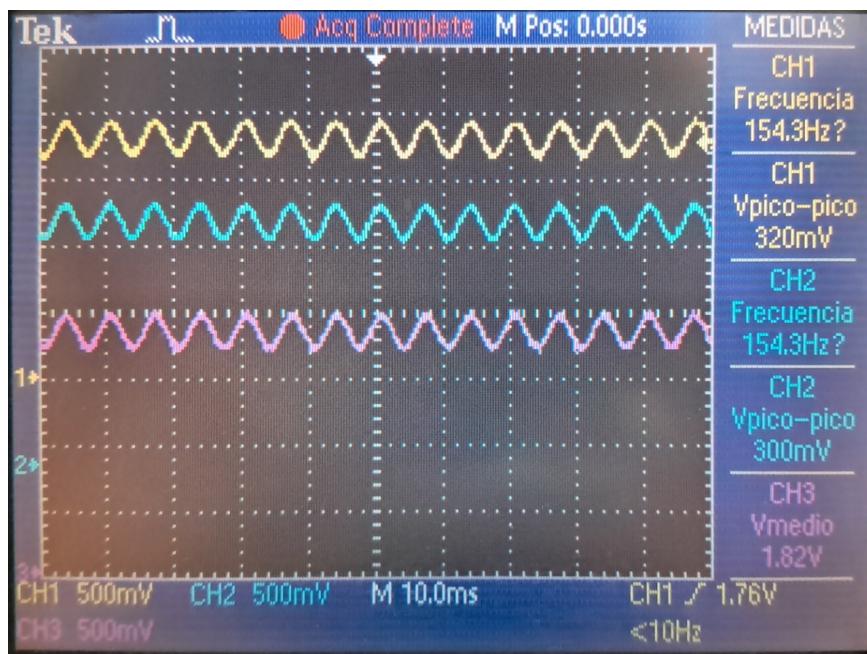


Fig. 5.2: Señal original medida desde osciloscopio

Lecturas con señal de 0.3 Vpp @ 150 Hz con placa Arduino Nano.
Muestreo a 500 Hz

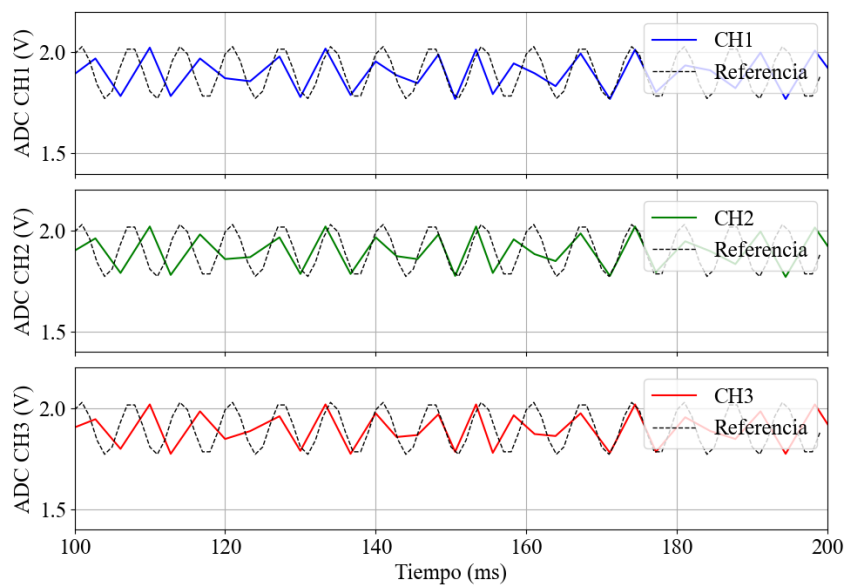


Fig. 5.3: Medición obtenida con un Arduino Nano de una señal con frecuencia de muestreo de 500 Hz

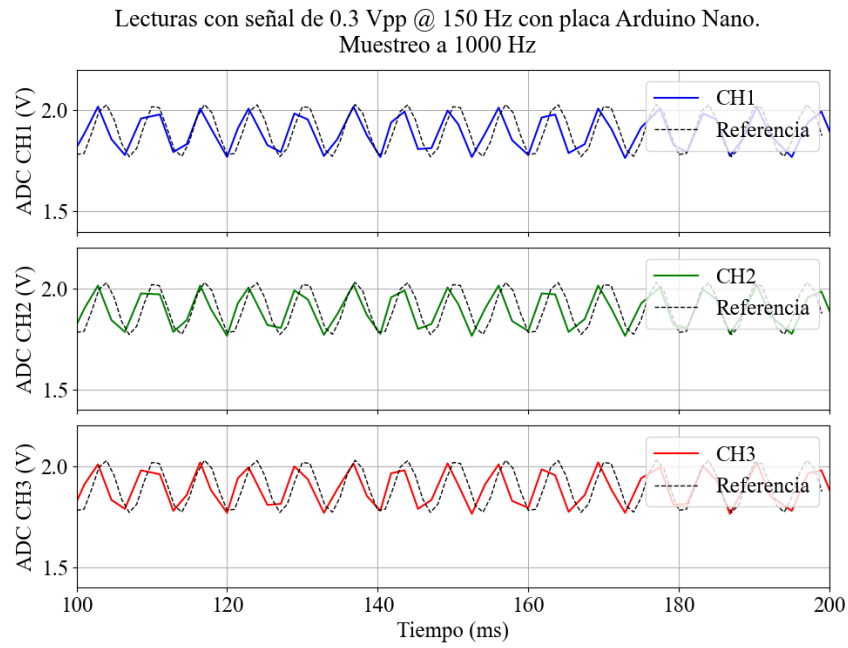


Fig. 5.4: Medición obtenida con un Arduino Nano de una señal con frecuencia de muestreo de 1 kHz

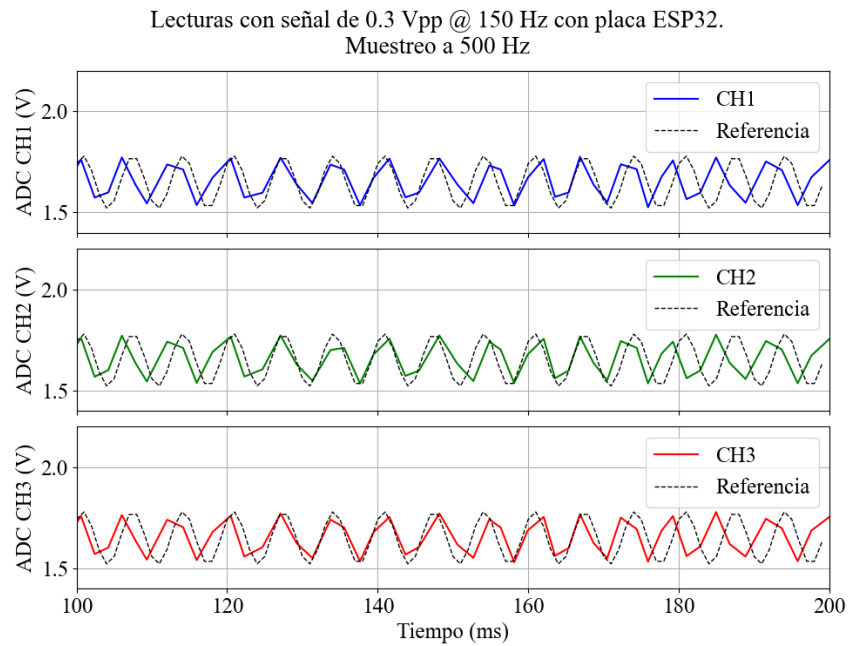


Fig. 5.5: Medición obtenida con una ESP32 de una señal con frecuencia de muestreo de 500 Hz

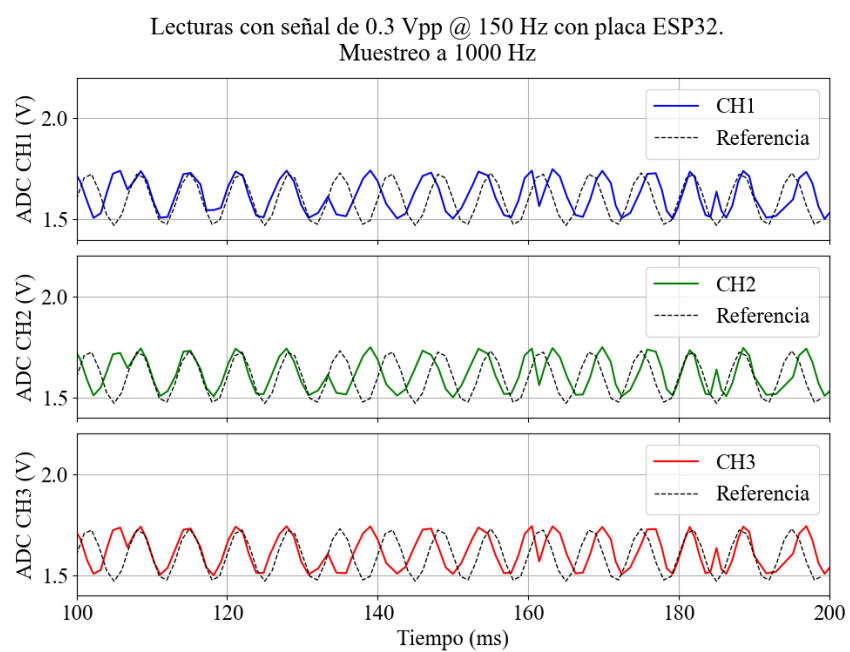


Fig. 5.6: Medición obtenida con una ESP32 de una señal con frecuencia de muestreo de 1 kHz

6. HARDWARE USADO

En los experimentos con los sensores se usa un microcontrolador, un computador portátil con Python instalado, los sensores Analog EMG Sensor de OYMotion, compuestos por una placa con el amplificador y otra con los electrodos. En lo que refiere a lo portable por el usuario, se diseña una parte montada en el antebrazo para sostener los electrodos conectada a una protoboard con un botón.

6.1. *Sensores Gravity Analog EMG de OyMotion*

Estos sensores son de electromiografía superficial y se componen de dos placas conectadas entre sí por un cable jack de 3,5mm. Una de las placas es el electrodo seco, compuesto de 3 placas metálicas que van en contacto directo con la piel. La otra placa es la amplificadora que aplica una ganancia de 1000 con un amplificador operacional de instrumentación e incluye también regulación de voltaje. Como salida, la señal retornada es análoga con un offset de 1,5 V y con variaciones que pueden llegar hasta los 3 V.

Los planos de los sensores se presentan en las figuras anexas A.1 y A.2, donde están las dimensiones de interés para manufacturar las cajas contenedoras del brazalete.

6.2. *Brazalete con sensores*

El brazalete con sensores está compuesto de una cinta elástica, que viene como parte del kit de los sensores y cajas contenedoras de placas para asegurar la posición

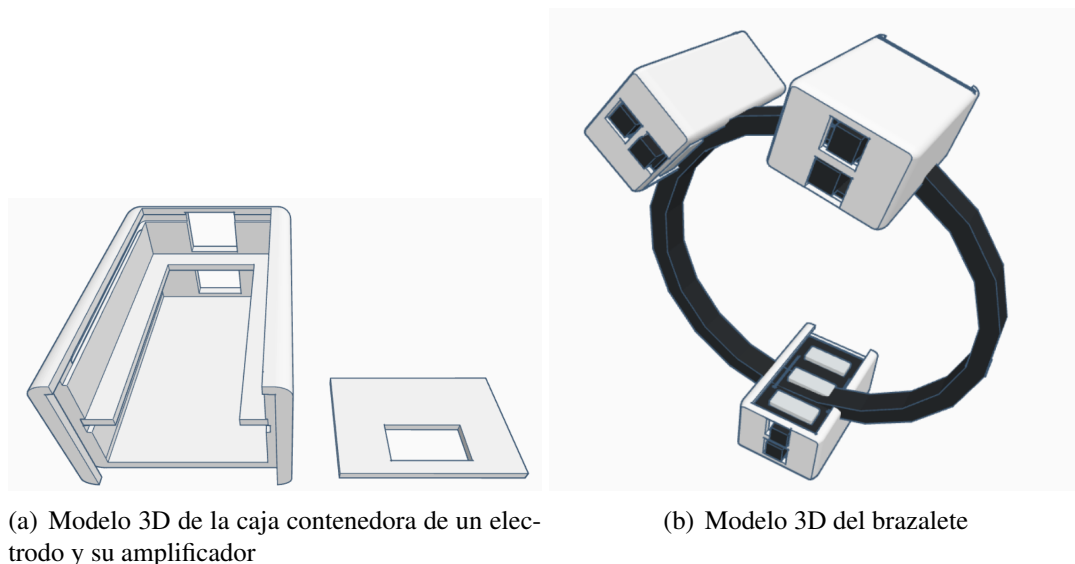


Fig. 6.1: Modelos 3D de las partes del brazalete

de los electrodos y amplificadores. En primera instancia se trabaja con tres electrodos representando a un canal cada uno, pero el diseño soporta hasta ocho canales dispuestos de forma radial en el antebrazo. La conexión entre los electrodos y las placas amplificadoras se hace usando los cables de 3,5mm incluidos en cada kit. El diseño de las cajas de los electrodos, así como el brazalete final, se presentan en la figura 6.1.

6.3. Circuito de adquisición

Se arma un circuito simple en una protoboard en el que se conectan los sensores, la placa microcontroladora y un pulsador que debe ser accionado por el paciente cada vez que ejecute un gesto. Las conexiones se presentan en la figura 6.2. La frecuencia de muestreo con la que opera el microcontrolador es de 1 kHz dado que las frecuencias de interés llegan hasta los 150 Hz.

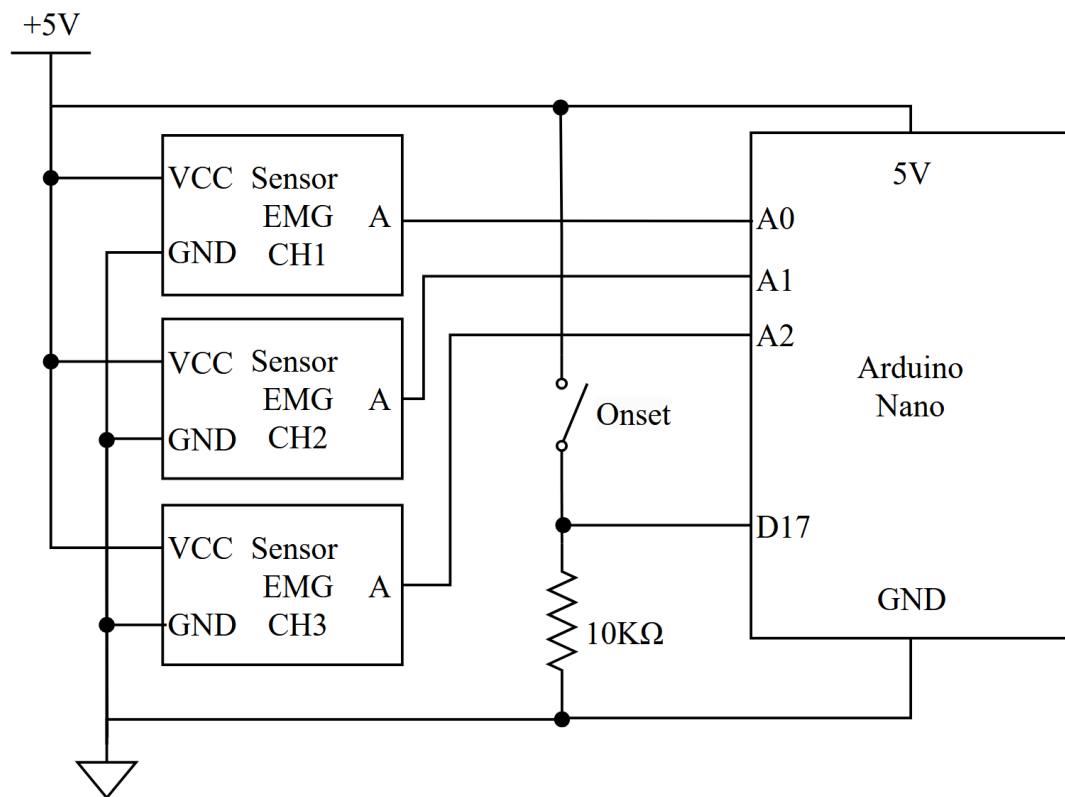


Fig. 6.2: Circuito de adquisición de señal y alimentación de sensores

7. CARACTERÍSTICAS DE INTERÉS DE LAS SEÑALES

El procesamiento de las señales contempla un análisis en cuanto a su amplitud, frecuencia, valor cuadrático medio (RMS) y relación señal-ruido (SNR). Para este trabajo en particular es de interés obtener la señal sin envolvente para el análisis de ruido y frecuencia.

7.1. *Estimación del espectro*

La estimación del espectro de las señales se hace por medio del método de Welch, que aplica una ventana a cada segmento de datos y calcula la transformada de Fourier de cada una. Finalmente, se promedian los resultados, lo que reduce la varianza de la estimación final en comparación con otros métodos como FFT [55]. Para ello, se hace uso de la biblioteca `welch` de SciPy.

7.2. *Cálculo de RMS y SNR*

El cálculo del valor cuadrático medio o Root Medium Square (RMS), así como de la SNR se hacen por medio de Python con las funciones de la librería NumPy. En este caso es de interés porque permite estimar el valor efectivo y la energía de las señales captadas por los sensores.

Para calcular la SNR se toman mediciones del músculo en reposo y de los gestos de interés. Los registros en reposo se consideran como mediciones de ruido de fondo para efectos del cálculo de la SNR. Para mantener la consistencia de los datos y los cálculos, en cada día se toman registros de ruido para ser usados con los gestos

capturados en la misma jornada. Por otra parte, para la señal en sí se usa el botón de onset para demarcar la ventana donde los músculos correspondientes están contraídos y eliminar los periodos donde únicamente se mide ruido. Como se plantea en [56], el cálculo de la SNR en sí se hace con el RMS de la señal y el ruido usando la fórmula de la ecuación (7.1).

$$SNR = 20 \cdot \log_{10} \left(\frac{S_{rms}}{N_{rms}} \right), \quad (7.1)$$

donde S_{rms} y N_{rms} son el voltaje eficaz de la señal y del ruido, respectivamente.

Este cálculo se hace en todos los gestos capturados para obtener un SNR promedio por cada uno de ellos.

8. CÓDIGO Y SCRIPTS USADOS

La implementación de este proyecto considera dos partes. Por un lado se usa una placa Arduino con un código que retorna las lecturas de los ADCs utilizados y un botón adicional que se debe mantener pulsado mientras se hace el gesto. La placa se conecta hacia un computador portátil funcionando con su batería para mantener el circuito aislado del ruido de 50 Hz de la corriente doméstica chilena. Los datos medidos por cada canal se retornan por la interfaz serial para su captura usando Python con librerías para la base de datos, la visualización y el procesamiento digital de las señales.

8.1. *Código del microcontrolador*

Dentro de la placa Arduino se carga un código que retorna las lecturas de los ADC utilizados a una tasa que se puede definir por el usuario. En este trabajo se usan 3 entradas análogas para los sensores y una cuarta para el botón de onset, que se debe mantener pulsado mientras se ejecuta un gesto. Considerando que las frecuencias en las que operan los músculos predominan entre 20 y 150 Hz [56], y los resultados obtenidos en la parte de validación, se utiliza una frecuencia de muestreo de 1 kHz medida con la función `micros` tomando el tiempo con la placa Arduino.

La salida de este programa imprime por el monitor serial los valores para cada canal en bruto, además del estado del botón onset, con el siguiente formato:

```
<onset>, <CH1>, <CH2>, <CH3>
```

Es importante mencionar que, para asegurar la rapidez de operación del microcontrolador y evitar la pérdida de datos, se envían los datos brutos retornados por el ADC.

Como tiene una resolución de 10 bits y un voltaje máximo de 5 V, los valores pueden variar entre 0 y 1023.

8.2. Código para el procesamiento de las señales

Para definir qué debe hacer el usuario, se genera una permutación aleatoria de 14 de los gestos para evitar una posible interferencia entre un gesto y otro por fatiga muscular. Los registros de reposo siempre se hacen al principio, y los de CVM al final. Cada gesto se ejecuta una vez, y durante cada ejecución se mantiene pulsado el botón de onset para demarcar los instantes en los que los músculos están activos.

8.2.1. Base de datos

Se genera una base de datos llamada `datos_gestos_3ch.db` que contiene varias tablas. Una de ellas es `raw`, donde se registran los datos brutos con los campos en la tabla 8.1, `norm` donde se registran los datos luego de ser filtrados cuyos campos se presentan en la tabla 8.2 normalizados, y `fft` donde se almacenan los resultados de los cálculos de FFT y SNR, en los campos de 8.3.

Campo	Tipo	Descripción
<code>id</code>	<code>int</code>	Identificador único autoincremental para cada entrada
<code>gesto_id</code>	<code>int</code>	Identificador único para cada gesto. Útil para diferenciar distintas instancias del mismo gesto
<code>sesion_id</code>	<code>int</code>	Número de la sesión en que se registró el gesto
<code>onset</code>	<code>int</code>	Indicador de si se está ejecutando el gesto. 1 para indicar que está en ejecución
<code>nombre_gesto</code>	<code>text</code>	Nombre del gesto hecho
<code>fs</code>	<code>int</code>	Frecuencia de muestreo en Hertz
<code>fecha</code>	<code>text</code>	Fecha en la que se hizo la captura, YYYY-MM-DD HH:MM:SS
<code>CHX</code>	<code>int</code>	Valor recibido para el canal X sin pasar por filtros

Tab. 8.1: Categorías dentro de la tabla raw

Campo	Tipo	Descripción
gesto_id	int	Identificador único para cada gesto. Útil para diferenciar distintas instancias del mismo gesto
sesion_id	int	ID de la sesión en la que se hizo la captura
nombre_gesto	text	Nombre del gesto hecho
fecha	text	Fecha en la que se hizo la captura, YYYY-MM-DD HH:MM:SS
fs	int	Frecuencia de muestreo en Hertz
fc	int	Frecuencia de corte del filtro
onset	int	Indicador de si se está ejecutando el gesto. 1 para indicar que está en ejecución
chX_norm	real	Valor del canal X normalizado respecto a la CVM correspondiente
chX_env_fil	real	Valor de la envolvente post filtrado del canal X

Tab. 8.2: Categorías dentro de la tabla norm

Campo	Tipo	Descripción
gesto_id	int	Identificador único para cada gesto. Útil para diferenciar distintas instancias del mismo gesto
sesion_id	int	ID de la sesión en la que se hizo la captura
nombre_gesto	text	Nombre del gesto hecho
fecha	text	Fecha en la que se hizo la captura, YYYY-MM-DD HH:MM:SS
fs	int	Frecuencia de muestreo en Hertz
fc	int	Frecuencia de corte del filtro
chX_fft	real	FFT de la señal en el canal X
chX_rms_senal	real	RMS de la señal en el canal X
chX_rms_ruido	real	RMS del ruido en el canal X
chX_SNR	real	SNR del canal X

Tab. 8.3: Categorías dentro de la tabla fft

8.2.2. Scripts generados

En este proyecto se generan scripts nuevos y se modifican algunos ya existentes para llevar a cabo la caracterización

Captura y visualización de datos

La captura de datos se hace con el script `lectura_3ch_rawEMG.py`, donde se le muestran al usuario los gestos registrados hasta el momento y se le solicita registrar el número de la sesión en que se están haciendo las capturas y el nombre del gesto a

capturar. Los resultados se registran en la tabla `raw` en sus correspondientes campos. Este código lee directamente desde el puerto serial configurado y registra los valores del onset y de los 3 canales, con el formato anteriormente expuesto. Se entregan también scripts auxiliares como `consultar_gestos.py` para desplegar por consola los datos almacenados en la base de datos, y `graficar_datos_3ch.py` para visualizar los 3 canales de un gesto a especificar por el usuario en base a su ID de gesto, también desplegada por consola.

Normalización de las señales

Habiendo obtenido las mediciones de los sensores y poblado la base de datos, el siguiente paso es medir el desempeño del sensor escogido. Para ello se utilizan las funciones del programa propuesto en [42] para filtrar, obtener la envolvente y normalizar las señales, llamado `emg_cvm_norm_sql.py` en este proyecto. Se modificó de tal forma que pueda conectarse a la base de datos, agregando las funciones tanto de consultar los datos brutos en `raw` como registrar los resultados obtenidos a `norm`.

Cálculo del espectro, RMS y SNR

Para calcular el espectro de las señales en bruto se usó el script `welch_datos_3ch.py` en el que se hacen consultas a la tabla de datos brutos, se elimina la componente continua y se aplica el método de Welch con la biblioteca SciPy en los 3 canales para posteriormente graficarlo.

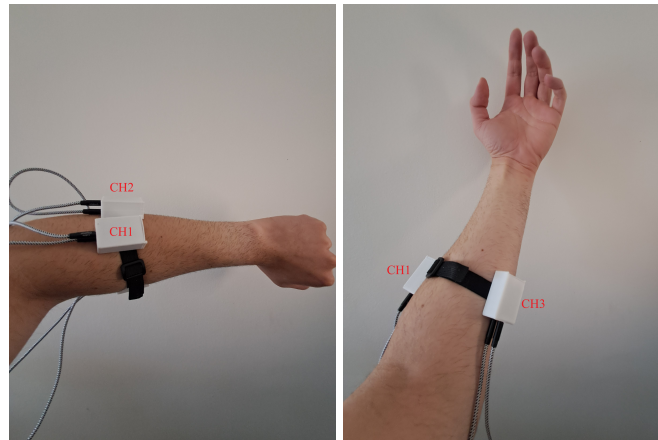
Para el cálculo de la RMS y SNR se generó el script `generar_tabla_fft_gestos.py`, en el que se genera la tabla `fft`, se hacen consultas a la tabla `norm` y se hacen los cálculos para registrarlos en la tabla `fft`. Se genera también el script auxiliar `calc_stddev_snr.py` para calcular la desviación estándar de la SNR obtenida.

9. PREPARACIÓN EXPERIMENTAL

Los sensores fueron dispuestos en el antebrazo sobre los músculos que están más relacionados con los gestos usados. Se usaron 3 canales siguiendo la lógica descrita en [54]. El canal 1 fue puesto sobre el músculo flexor radial del carpo. El canal 2 fue puesto sobre los músculos extensor común de dedos y extensor cubital del carpo, ubicables al extender la muñeca y los dedos. Finalmente, el canal 3 se ubicó en el flexor común superficial de los dedos, que está principalmente asociado a movimientos de flexión de dedos y muñeca. El brazalete con la posición de los canales se presenta en la figura 9.1.

Antes de poner el brazalete en el paciente, se limpiaron las zonas de contacto y los electrodos con alcohol desnaturalizado de 95°. Durante cada sesión, el sujeto se encuentra descansado, sentado con el codo apoyado en un escritorio y levantando el antebrazo solamente a la hora de ejecutar un gesto. Luego de haber colocado los sensores, se usa el Serial Plotter de Arduino para monitorear las señales recibidas desde el puerto serial para cada canal. Para asegurar una correcta captura de los datos, se debe revisar que la señal no varíe mucho y no sea muy ruidosa. En el caso de recibir lecturas brutas desde un Arduino Nano, es preferible que el ruido captado en reposo no sea mucho mayor a 10 unidades de su ADC de 10 bits. Si se experimentan ruidos, se deben ir ajustando los sensores moviéndolos ligeramente y asegurándose que estén bien presionados con la piel. También es recomendable esperar unos minutos para que mejore la conductividad de los electrodos secos.

A modo de comparación para ver el impacto de la velloidad de los brazos del paciente, las primeras sesiones se hacen sin afeitar sus antebrazos. Las últimas sesiones se hacen con los brazos posteriormente afeitados, respetando la posición de los senso-



(a) Posición de los canales 1 y 2 (b) Posición del canal 3

Fig. 9.1: Ubicación de los sensores en el antebrazo

res y manteniendo la rutina de limpieza para maximizar el contacto entre electrodo y piel.

Durante la ejecución de cada gesto, se ejecuta el script en Python para ir registrando los datos y, al empezar a hacer el gesto solicitado, se mantiene pulsado el botón asociado a *onset* ubicado en la protoboard con los componentes. Cuando se haya mantenido el gesto por al menos 1 segundo, se suelta el botón de onset y se relaja el antebrazo, dando entre 10 y 30 segundos luego de cada gesto para descansar el brazo. Se toman mediciones a lo largo de una semana y se permuta el orden de los gestos hechos para evitar posibles interferencias asociadas al cansancio de los músculos.

10. RESULTADOS

A continuación se presentan los resultados más representativos de las mediciones y cálculos hechos.

10.1. Mediciones brutas por gesto

Las mediciones en su estado más bruto se presentan en las figuras 10.1 y 10.2. Estas representan el retorno de los sensores tal como se percibe desde la placa adquisidora, sin aplicar filtros ni detección de envolvente. Las señales estuvieron centradas en 1,5 V y presentaron valores peak-to-peak entre 0,6 V y 3 V, tal como se presenta en la tabla 10.1. De esto se puede extraer que la apertura de manos, hacer un puño, y los movimientos de flexión y extensión son los que más señal generan.

10.2. Normalización de señales

La normalización se hace con la propuesta hecha en [42], donde se usan las contracciones máximas voluntarias como referencia para normalizar las señales. Dada la gran cantidad de volumen de gestos triplicada por la cantidad de canales usados, se escoge el puño como referencia dado su uso común y facilidad de detección con los sensores, obteniéndose los gráficos de la figura 10.3.

10.3. Cálculo de RMS y SNR

Los cálculos de RMS y SNR se hacen con la envolvente de las señales, usando las de reposo como ruido de la correspondiente sesión y canal. Se usan los datos

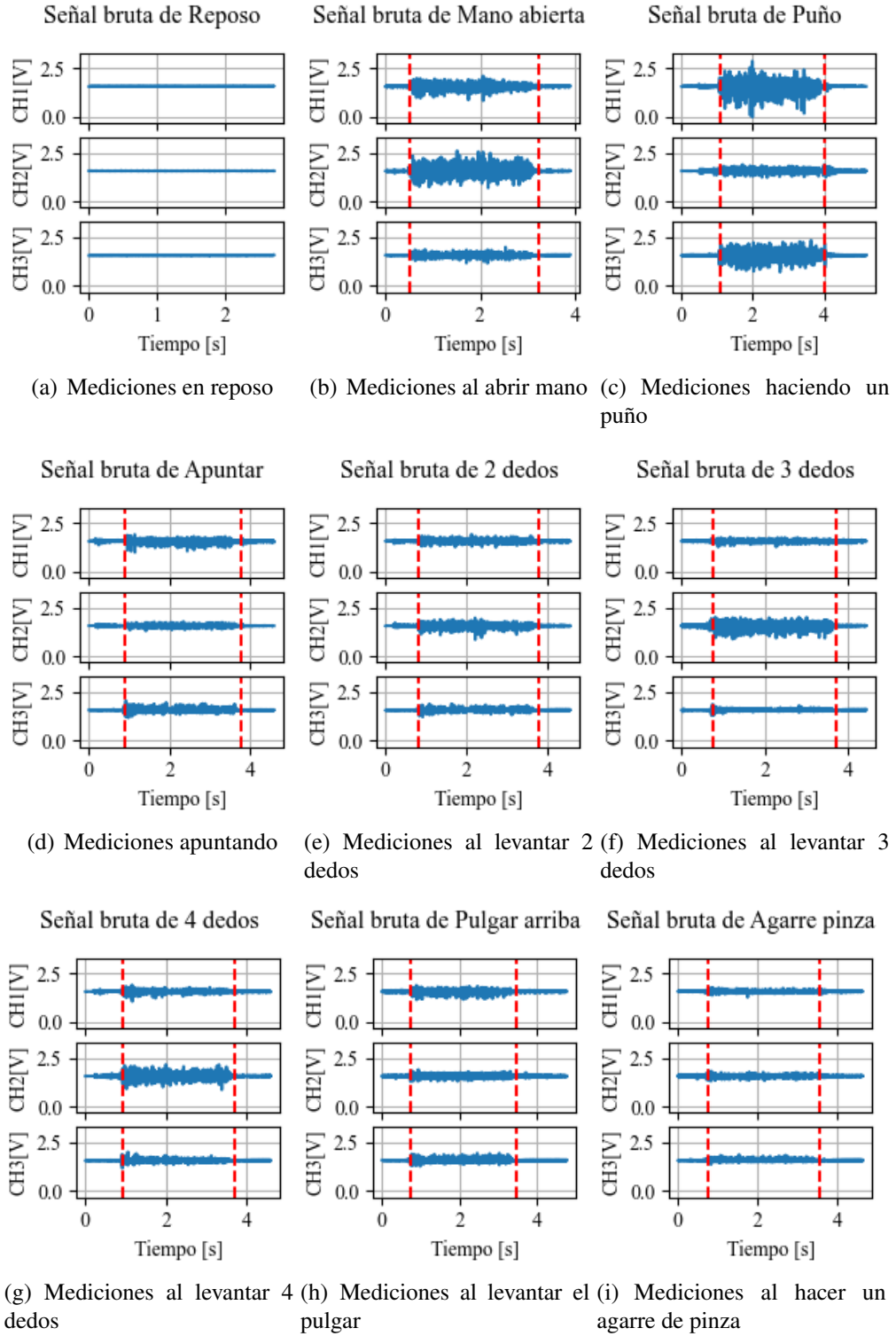
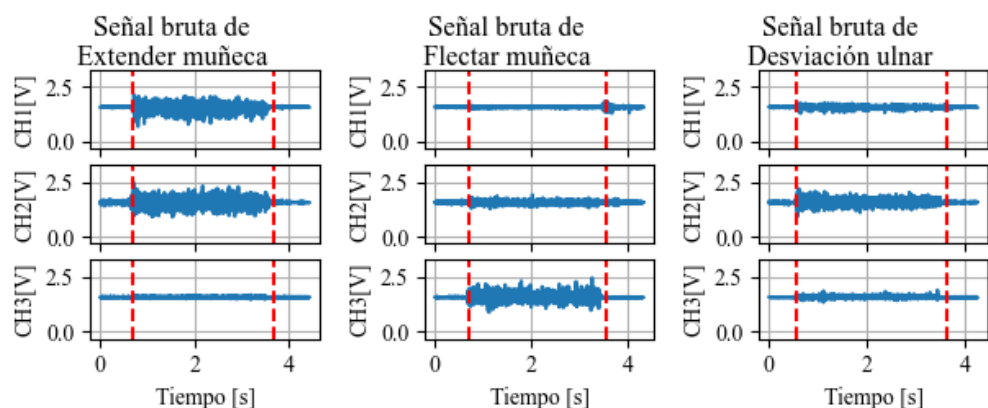
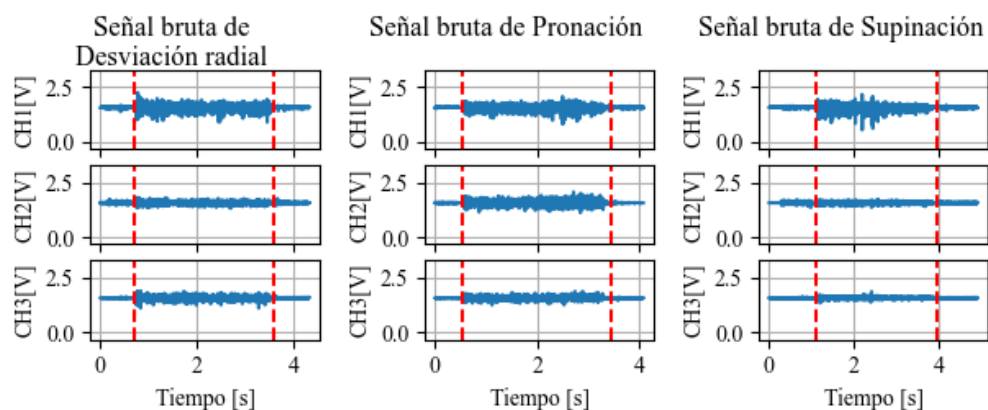


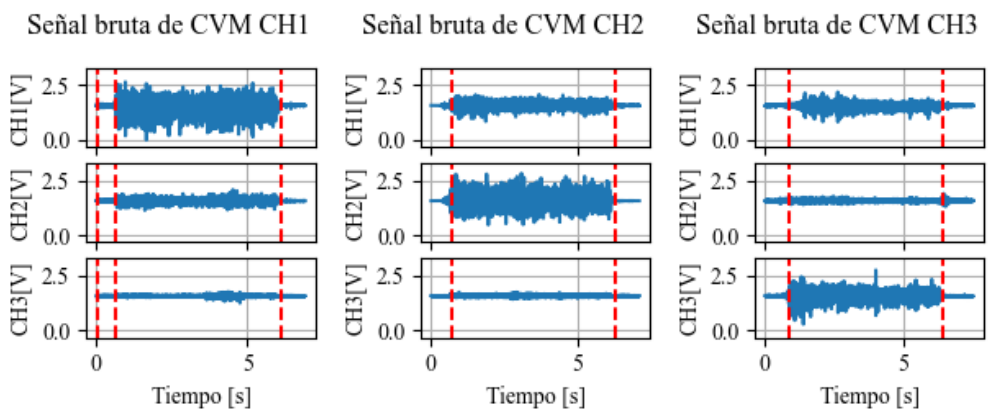
Fig. 10.1: Primer set de capturas brutas hechas. Las líneas verticales representan al inicio y fin de la ejecución del gesto



(a) Mediciones al extender muñeca (b) Mediciones al flexar la muñeca (c) Mediciones al mover muñeca hacia meñique

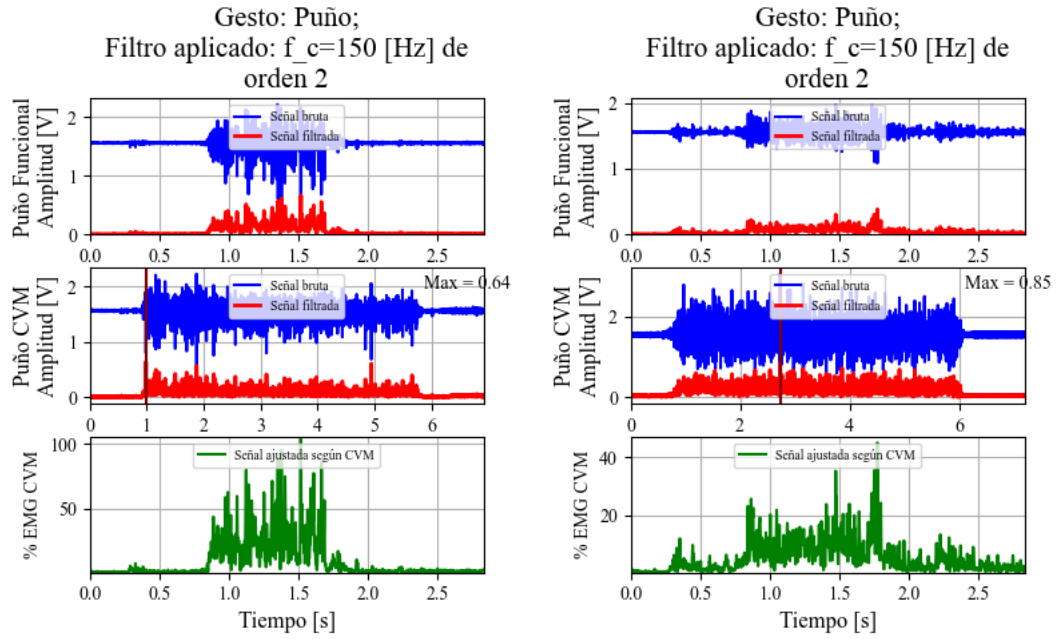


(d) Mediciones al mover muñeca hacia pulgar (e) Mediciones al girar muñeca hacia adelante (f) Mediciones al girar muñeca hacia uno



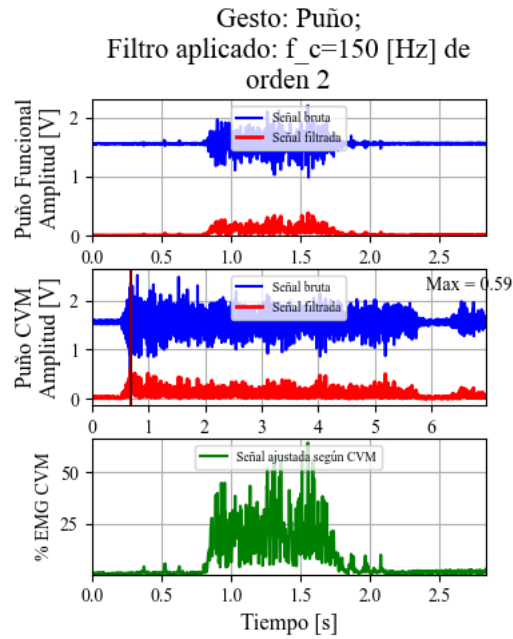
(g) Mediciones al tensar el músculo flexor radial del carpo (h) Mediciones al tensar el extensor común de dedos (i) Mediciones al contraer el tensor común superficial de los dedos

Fig. 10.2: Segundo set de capturas brutas. Las líneas verticales representan al inicio y fin de la ejecución del gesto



(a) CH1 normalizado

(b) CH2 normalizado



(c) CH3 normalizado

Fig. 10.3: Normalización del gesto "Puño" usando la CVM y un filtro de segundo orden a 150 Hz

Nombre del gesto	Amplitud CH1 [Vpp]	Amplitud CH2 [Vpp]	Amplitud CH3 [Vpp]
2 dedos	0.94	1.32	0.68
3 dedos	0.66	1.72	0.82
4 dedos	0.93	1.67	0.89
Agarre pinza	0.54	1.42	0.62
Apuntar	1.13	0.74	1.3
CVM CH1	2.64	1.28	0.86
CVM CH2	2.47	2.75	0.51
CVM CH3	1.36	1.57	3.03
Desviación radial	1.47	0.68	0.87
Desviación ulnar	0.65	1.74	0.74
Extender muñeca	1.96	2.2	0.43
Flectar muñeca	0.63	1.04	1.95
Mano abierta	1.27	2.17	0.89
Pronación	1.3	1.22	1.05
Pulgar arriba	0.81	0.9	0.96
Puño	2.86	0.91	2.11
Reposo	0.06	0.08	0.11
Supinación	1.69	0.72	1.8

Tab. 10.1: Amplitud por gesto

correspondientes a las 7 sesiones de captura. Los RMS calculados se concentran en la tabla 10.2 y los SNR calculados en la tabla 10.3.

De los RMS obtenidos se obtiene que las señales más energéticas son las de las contracciones voluntarias musculares de sus respectivos canales. Estos valores pueden dar una idea de cuál es el voltaje efectivo máximo que se puede captar por sensor y por canal. Por otra parte, los menores son los del brazo en reposo y se pueden asociar al ruido del canal, lo que da paso a calcular la SNR y ver qué tan bien se distingue la señal del músculo con respecto al ruido del mismo canal.

Del resto de los gestos, los más notables son los relacionados a movimientos de puño y muñeca para el canal 1, apertura de mano y movimientos de muñeca para el canal 2, y movimientos de flexión de muñeca y puño para el canal 3. Los gestos que involucraron movimientos más sutiles como agarre de pinza, levantar de a pocos dedos y giros de muñeca son los que retornaron menor señal en general.

En cuanto a qué tan discernible es la señal asociada a cada gesto, los que involucran movimientos de muñeca, abrir y cerrar la mano son los que mejor se distinguen

Nombre del gesto	RMS CH1 [V]	RMS CH2 [V]	RMS CH3 [V]
2 dedos	0.05	0.09	0.05
3 dedos	0.04	0.12	0.03
4 dedos	0.05	0.11	0.05
Agarre pinza	0.03	0.08	0.03
Apuntar	0.06	0.05	0.07
CVM CH1	0.18	0.07	0.03
CVM CH2	0.12	0.24	0.03
CVM CH3	0.04	0.04	0.17
Desviación radial	0.09	0.05	0.06
Desviación ulnar	0.04	0.13	0.04
Extender muñeca	0.13	0.16	0.02
Flectar muñeca	0.02	0.06	0.13
Mano abierta	0.08	0.18	0.05
Pronación	0.08	0.09	0.06
Pulgar arriba	0.05	0.06	0.05
Puño	0.17	0.07	0.16
Reposo	0.005	0.01	0.01
Supinación	0.06	0.04	0.04

Tab. 10.2: Valores promedio de RMS

respecto al ruido por sus valores de SNR más altos. Además, como todos los gestos retornaron un SNR mayor a cero, significa que todos los casos la señal es lo suficientemente discernible como para distinguirlos del ruido. En el caso particular del reposo se obtiene un SNR de 0 porque fue usado como referencia de ruido.

En la tabla 10.4 se concentran los valores de desviación estándar de la SNR a lo largo de las distintas sesiones de captura. Acá se evidencia que hay alta variabilidad en la forma que se reciben las señales entre las sesiones, lo que puede ser provocado por pequeñas variaciones en cómo se coloca el sensor en el antebrazo, el nivel de tensión al que puede llegar el músculo, el cansancio, entre otros factores más.

Nombre del gesto	SNR promedio CH1 [dB]	SNR promedio CH2 [dB]	SNR promedio CH3 [dB]
2 dedos	22.28	23.02	15.13
3 dedos	20.64	26.57	12.48
4 dedos	22.33	25.25	15.35
Agarre pinza	19.57	21.97	12.5
Apuntar	25.07	18.13	18.57
CVM CH1	33.02	20.28	9.98
CVM CH2	29.58	31.32	10.93
CVM CH3	17.08	15.74	25.87
Desviación radial	28.29	17.48	17.3
Desviación ulnar	22.05	26.42	13.79
Extender muñeca	31.6	28.22	8.82
Flectar muñeca	16.25	19.53	24.56
Mano abierta	26.88	29.5	16.87
Pronación	27.01	23.28	17.37
Pulgar arriba	22.9	20.25	16.58
Puño	34.17	21.59	25.88
Reposo	0.0	0.0	0.0
Supinación	24.61	16.96	13.34

Tab. 10.3: Valores promedio de SNR

Nombre del gesto	SNR CH1 [dB]	SNR CH2 [dB]	SNR CH3 [dB]
2 dedos	6.23	7.68	3.10
3 dedos	6.35	7.06	3.69
4 dedos	7.89	8.35	3.87
Agarre pinza	6.15	7.85	4.18
Apuntar	6.15	7.42	3.15
CVM CH1	7.79	8.65	5.71
CVM CH2	7.05	7.49	4.80
CVM CH3	8.64	7.30	5.65
Desviación radial	5.04	7.57	3.78
Desviación ulnar	6.01	7.64	3.18
Extender muñeca	5.86	9.45	4.52
Flectar muñeca	5.94	8.37	3.25
Mano abierta	5.91	8.31	4.44
Pronación	5.97	7.96	3.65
Pulgar arriba	4.97	7.50	3.48
Puño	5.71	7.26	4.45
Reposo	0.00	0.00	0.00
Supinación	6.23	7.31	4.79

Tab. 10.4: Desviación estándar de SNR por canal

10.4. *Estimación del espectro usando Welch*

El cálculo hecho a partir de las señales brutas se presenta en las figuras 10.4 y 10.5. Para efectuarlo se eliminó la componente continua de los registros y se reescaló para trabajar con valores de voltaje en lugar de los del ADC. Se calculó usando los valores por defecto, con segmentos de 256 muestras por cada uno y 50 % de superposición. De las figuras se puede extraer que la mayor actividad muscular registrada se encuentra en la banda de los 20 a 200 Hz, lo que coincide con lo encontrado con la literatura. Se obtiene también que, en las señales de reposo, el ruido está cercano a los 50 Hz, lo que puede ser atribuido a ruido ambiental, procesos corporales y a pérdidas de contacto efectivo con la piel por el menor tamaño del músculo al estar relajado. Al ver los movimientos de contracción máxima voluntaria, se puede observar el mismo ruido visto en el registro de reposo, como también se evidencia una cierta interferencia entre los canales 1 y 2.

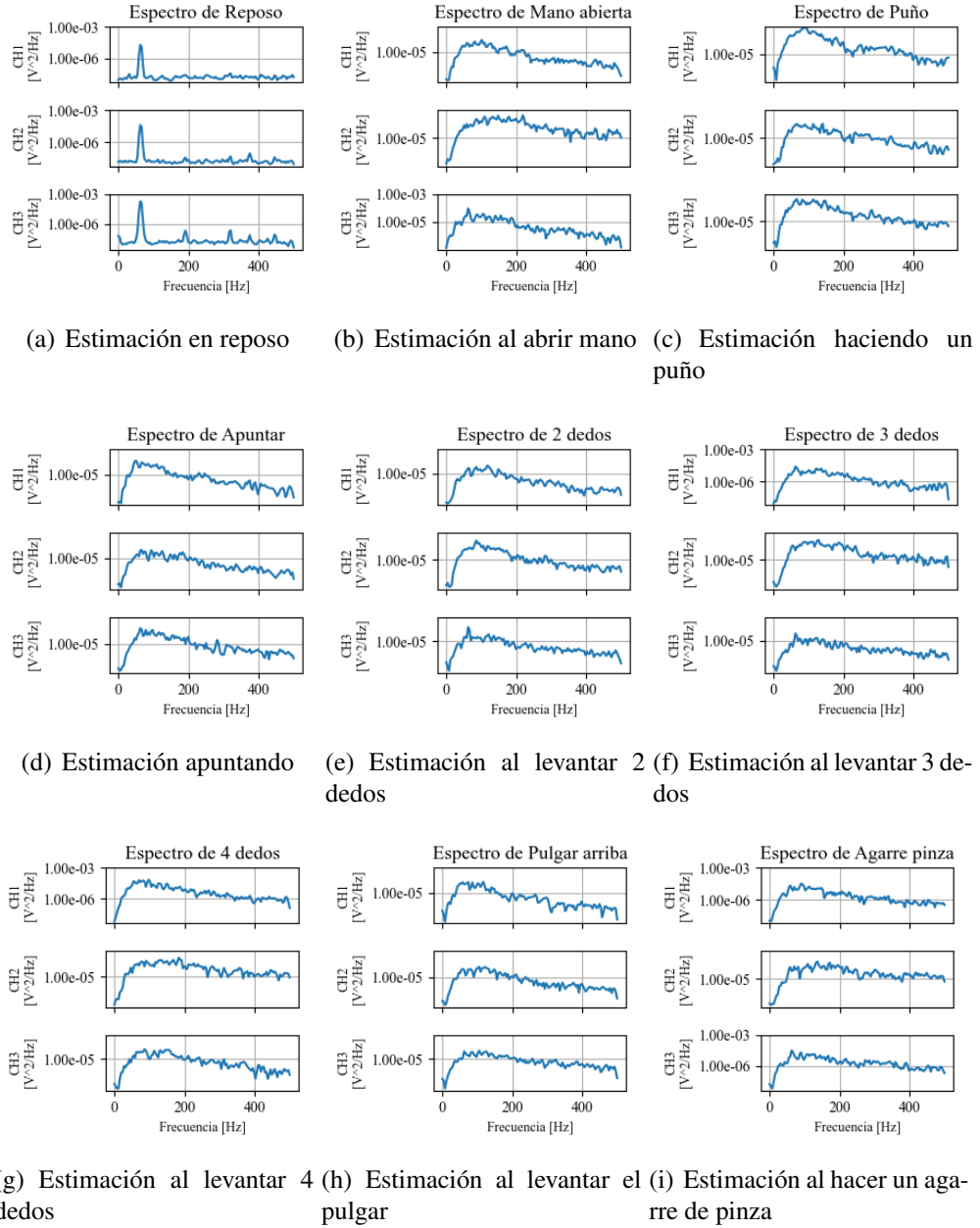
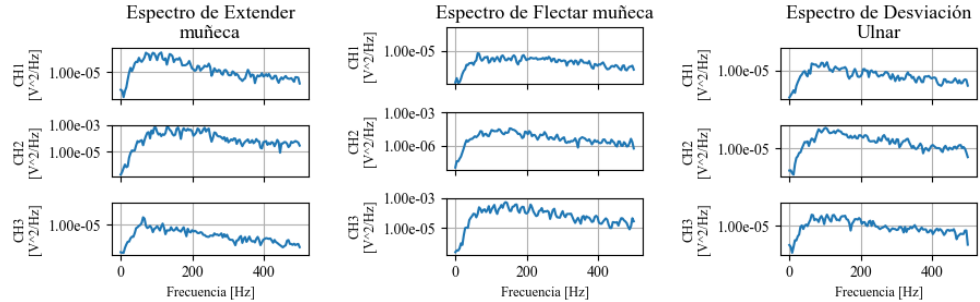
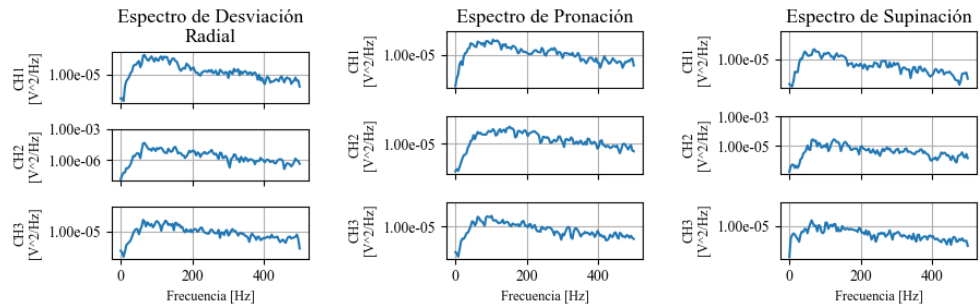


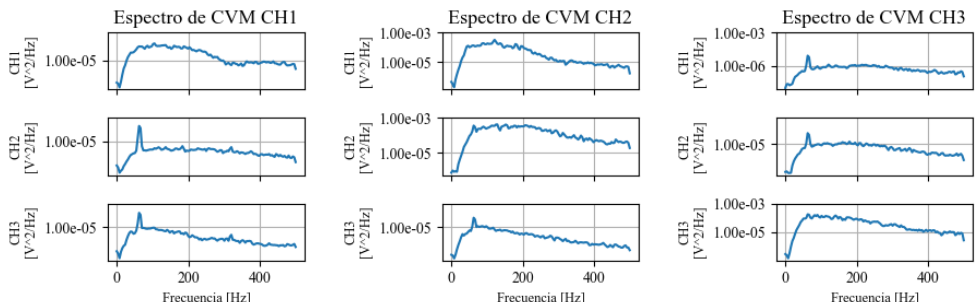
Fig. 10.4: Primer set de espectros calculados con las señales sin filtrar



(a) Estimación al extender la muñeca (b) Estimación al flexar la muñeca (c) Estimación al mover muñeca hacia meñique



(d) Estimación al mover muñeca hacia pulgar (e) Estimación al girar muñeca hacia adelante (f) Estimación al girar muñeca hacia uno



(g) Estimación al tensar el músculo flexor radial del carpo (h) Estimación al tensar el extensor común de dedos (i) Estimación al contraer el tensor común superficial de los dedos

Fig. 10.5: Segundo set de espectros calculados con las señales sin filtrar

11. CONCLUSIONES Y TRABAJO FUTURO

En este trabajo se propone una metodología para caracterizar un sensor sEMG comercialmente disponible, un diseño con el que se pueden montar dichos sensores en el antebrazo de un paciente, las posiciones donde se pueden colocar para detectar un abanico de movimientos de mano y una serie de scripts y funciones que permiten registrar las capturas y analizarlas posteriormente.

Al ejecutar los gestos y analizar posteriormente sus resultados se obtiene que, dada la posición de los sensores, los gestos finos retornaron menores niveles que los gestos de muñeca. En particular la apertura y cierre de mano, como también los movimientos de extensión y flexión de muñeca fueron los que mejor pudieron ser detectados. Sin embargo, los movimientos de giro como pronación, supinación, desviación ulnar y radial no obtuvieron tanto éxito. Por otra parte, los levantamientos de dedos obtuvieron espectros parecidos a los de mano abierta con la diferencia que los de mano completa tuvieron mayor amplitud en el espectro.

Respecto a la normalización propuesta en [42] se puede decir que es una herramienta útil a la hora de ver la excitación de los músculos al ejecutar gestos porque usar un punto de referencia, como la contracción máxima por canal, puede ayudar a subsanar eventualidades e imperfecciones en la instalación o impedancia de los sensores. Esto también evidencia el hecho de que no todos los canales van a estar operando al mismo nivel de voltaje, porque este valor también se ve afectado por la profundidad de los músculos hacia el brazo. Como se usan sensores superficiales, es esperable que los músculos más profundos y pequeños retornen un voltaje menor, por lo que podría considerarse una buena práctica acotar los límites que pueden alcanzar los músculos y empezar a trabajar desde ahí.

Cuando se implementó el código de prueba entregado por el proveedor [57] en la placa Arduino se obtuvo una envolvente cuyo valor escapaba de los rangos de voltaje esperables. Esto ocurría porque se estimaba la señal por medio de pasarla por un filtro pasabajos y elevarla al cuadrado, pero en la práctica esta aproximación no retornaba una señal con la que se pudiera trabajar.

El uso de una base de datos como forma de registrar datos demostró ser útil a la hora de manejar grandes cantidades de registros porque permitió usar varias formas de indexar los gestos que fueron tomando, así como también permitió hacer operaciones haciendo consultas a la base de datos en lugar de generar scripts.

Uno de los desafíos encontrados a la hora de registrar gestos se dio durante el uso de Python y la base de datos. En una primera toma de muestras se hicieron registros de manera constante hacia la base de datos, pero haciendo un análisis posterior se reveló que aparecía un retraso de varios segundos y, al detener la ejecución del script, se perdía alrededor del 80 % de las muestras porque estaban en un buffer que se borraba al enviar el comando de interrupción. La solución a este problema fue guardar los datos directamente en la memoria usando una lista dentro del mismo Python y luego hacer registros periódicamente en la base de datos. Con esto, se resolvieron varias pérdidas de muestras y deformaciones en frecuencia de la señal.

Durante la adquisición de datos se manufacturaron placas perforadas como plataformas para alimentar los sensores y conectarlos al microcontrolador. Sin embargo, al visualizar las señales recibidas se experimentaron interferencias entre canales e inducciones de voltaje inesperadas, lo que evidenció que la sensibilidad de los componentes y las frecuencias en las que se está trabajando requieren que exista mayor énfasis en la aislación y distanciamiento de las pistas. Al usar una protoboard con cables para conectar los componentes se redujeron en gran medida los efectos indeseados, lo que confirma esta idea dado que los cables estaban separados y sugiere que una futura implementación de este circuito se haga con una PCB impresa. Dicha futura implementación debe considerar suficiente espaciado entre pistas y buen aislamiento, sobre todo si se diseña para montarse en el antebrazo de un usuario.

Como trabajo futuro se plantea incrementar la cantidad de mediciones registradas en la base de datos siguiendo la misma metodología planteada y entrenar una red neuronal para aplicar reconocimiento de gestos usando el equipo diseñado.

Bibliografía

- [1] OpenBionics, “Openbionics.” [en línea] <<https://openbionics.com/>> [consulta: 11 mayo 2024].
- [2] M. Gomez-Correa and D. Cruz-Ortiz, “Low-cost wearable band sensors of surface electromyography for detecting hand movements,” *Sensors*, vol. 22, p. 5931, 2022.
- [3] N. Inc., “Neurosky.” [en línea] <<https://neurosky.com/>> [consulta: 8 junio 2024].
- [4] A. Marsh, “The cold war arms race over prosthetic arms,” *IEEE Spectrum*, 2023. [en línea] <<https://spectrum.ieee.org/prosthetic-arm>> [consulta: 11 mayo 2024].
- [5] E. Kwek and M. Choi, “Is a prosthetic arm customized prada? a critical perspective on the social aspects of prosthetic arms,” *Disability & Society*, vol. 31, no. 8, pp. 1144–1147, 2016.
- [6] e-NABLE, “Enabling the future.” [en línea] <<http://enablingthefuture.org/>> [consulta: 11 mayo 2024].
- [7] Fundación Prótesis 3D Chile, “Fundación prótesis 3d chile.” [en línea] <<https://www.fp3d.cl/>> [consulta: 08 mayo 2024].
- [8] K. J. Zuo and J. L. Olson, “The evolution of functional hand replacement: From iron prostheses to hand transplantation,” *Plastic Surgery*, vol. 22, no. 1, pp. 44–51, 2014.

-
- [9] P. Vanich, P. Tangpornprasert, and C. Virulsri, “Design of a single-dof prosthetic hand with practical maximum grip force and grasp speed for adls using dual-motor actuator,” *IEEE Robotics and Automation Letters*, vol. 8, pp. 1439–1446, March 2023.
- [10] Upper Limb Prosthetics. Hanger Clinic, “Upper limb prosthetics.” [en línea] <<https://hangerclinic.com/prosthetics/upper-limb/>> [consulta: 12 mayo 2024].
- [11] H. Basumatary and S. M. Hazarika, “State of the art in bionic hands,” *IEEE Transactions on Human-Machine Systems*, vol. 50, pp. 116–130, April 2020.
- [12] J. B. Y. Gloumakov and A. M. Dollar, “Trajectory control—an effective strategy for controlling multi-dof upper limb prosthetic devices,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 30, pp. 420–430, 2022.
- [13] A. Bahador, M. Yousefi, M. Marashi, and O. Bahador, “High accurate light-weight deep learning method for gesture recognition based on surface electromyography,” *Computer Methods and Programs in Biomedicine*, vol. 195, p. 105643, 2020.
- [14] A. Fougner, O. Stavdahl, P. J. Kyberd, Y. G. Losier, and P. A. Parker, “Control of upper limb prostheses: Terminology and proportional myoelectric control—a review,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 20, pp. 663–677, Sept. 2012.
- [15] N. V and Z. Y-P, “Controlling upper limb prostheses using sonomyography (smg): A review,” *Sensors*, vol. 23, no. 1885, 2023.
- [16] A. Fougner, M. Sæther, Ø. Stavdahl, P. J. Kyberd, and J. Blum, “Cancellation of force induced artifacts in surface emg using fsr measurements,” in *Proceedings of the 2008 MyoElectric Controls/Powered Prosthetics Symposium*, pp. 13–15, August 2008.

-
- [17] D. R. C. Puma, “A low-cost robotic hand prosthesis with apparent haptic sense controlled by electroencephalographic signals,” 2023.
- [18] M. I. Uygun, B. Recber, M. A. Celikli, and Y. Oniz, “An emg controlled bionic arm with machine learning,” in *2023 7th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, (Ankara, Turkiye), pp. 1–6, 2023.
- [19] J. S. Artal-Sevil, A. Acon, J. L. Montanes, and J. A. Dominguez, “Design of a low-cost robotic arm controlled by surface emg sensors,” in *2018 XIII Technologies Applied to Electronics Teaching Conference (TAEE)*, pp. 1–8, 2018.
- [20] Bionico.org, “La bionicohand.” [en línea] <<https://bionico.org/la-bionicohand/>> [consulta: 12 mayo 2024].
- [21] Ottobock, “Axonhook.” [en línea] <<https://www.ottobock.com/es-es/product/8E600>> [consulta: 12 mayo 2024].
- [22] A. Dynamics, “Introduction to the etd2.” [en línea] <<https://www.armdynamics.com/upper-limb-library/introduction-to-the-etd2>> [consulta: 12 mayo 2024].
- [23] Össur, “Introducción a i-limb.” [en línea] <<https://www.ossur.com/es-es/protesis/entrenamiento-de-extremidades-superiores/i-limb>> [consulta: 12 mayo 2024].
- [24] B. H. 2.0, “Thingiverse.” [en línea] <<https://www.thingiverse.com/thing:3000641>> [consulta: 12 mayo 2024].
- [25] A. Villoslada, “Project details for dextra.” [en línea] <<https://hackaday.io/project/9890-dextra/details>> [consulta: 12 mayo 2024].
- [26] Ottobock, “bebionic.” [en línea] <<https://www.ottobock.com/es-mx/product/8E70>> [consulta: 12 mayo 2024].

-
- [27] Ottobock, “Michelangelo.” [en línea] <<https://www.ottobock.com/es-es/product/8E500>> [consulta: 12 mayo 2024].
- [28] Ottobock, “Sensorhand speed.” [en línea] <<https://www.ottobock.com/es-es/product/8E3958>> [consulta: 12 mayo 2024].
- [29] V. Systems, “Vincentyoung3+.” [en línea] <<https://www.vincentsystems.de/en/vincent-young3>> [consulta: 12 mayo 2024].
- [30] Össur, “i-digits quantum.” [en línea] <<https://www.ossur.com/es-es/protésica/miembro-superior/i-digits-quantum>> [consulta: 12 mayo 2024].
- [31] A. Dynamics, “Taska cx.” [en línea] <<https://www.armdynamics.com/taska-cx>> [consulta: 12 mayo 2024].
- [32] Psyonic, “Ability hand system.” [en línea] <<https://www.spsco.com/ability-hand-system.html>> [consulta: 12 mayo 2024].
- [33] U. Tomorrow, “Truelimb prosthetic arm.” [en línea] <<https://www.unlimitedtomorrow.com/truelimb/>> [consulta: 12 mayo 2024].
- [34] M. Bionics, “Luke arm details.” [en línea] <<https://mobiusbionics.com/luke-arm/>> [consulta: 12 mayo 2024].
- [35] S. Group, “Split hooks.” [en línea] <<https://www.steepergroup.com/prosthetics/upper-limb-prosthetics/hands/split-hooks/>> [consulta: 12 mayo 2024].
- [36] M. Tavakoli, C. Benussi, P. A. Lopes, L. B. Osorio, and A. T. de Almeida, “Robust hand gesture recognition with a double channel surface emg wearable arm-band and svm classifier,” *Biomedical Signal Processing and Control*, vol. 46, pp. 121–130, 2018.

-
- [37] M. S. Al-Quraishi, I. Elamvazuthi, S. A. Daud, S. Parasuraman, and A. Borboni, “Eeg-based control for upper and lower limb exoskeletons and prostheses: A systematic review,” *Sensors*, vol. 18, no. 10, p. 3342, 2018.
- [38] EMOTIV, “Insight.” [en línea] <<https://www.emotiv.com/products/insight>> [consulta: 8 junio 2024].
- [39] M. Y. Ng, M. Pourmajidian, and N. A. Hamzaid, “Mechanomyography sensors for detection of muscle activities and fatigue during fes-evoked contraction,” in *2014 IEEE 19th International Functional Electrical Stimulation Society Annual Conference (IFESS)*, pp. 1–3, IEEE, 2014.
- [40] P. Konrad, “The abc of emg,” *A practical introduction to kinesiological electromyography*, vol. 1, no. 2005, pp. 30–5, 2005.
- [41] A. Technologies, *MyoWare Advanced Guide*, 2022.
- [42] O. Valencia, C. De La Fuente, R. Guzmán-Venegas, R. Salas, and A. Weinstein, “Propuesta de flujo de procesamiento utilizando python para ajustar la señal electromiográfica funcional a la contracción voluntaria máxima,” *Revista Kinesiología*, 2021.
- [43] M. A. Oskoei and H. Hu, “Myoelectric control systems—a survey,” *Biomedical Signal Processing and Control*, vol. 2, no. 4, pp. 275–294, 2007.
- [44] G. Yang, J. Deng, G. Pang, H. Zhang, J. Li, B. Deng, Z. Pang, J. Xu, M. Jiang, P. Liljeberg, *et al.*, “An iot-enabled stroke rehabilitation system based on smart wearable armband and machine learning,” *IEEE journal of translational engineering in health and medicine*, vol. 6, pp. 1–10, 2018.
- [45] U. Côté-Allard, G. Gagnon-Turcotte, F. Laviolette, and B. Gosselin, “A low-cost, wireless, 3-d-printed custom armband for semg hand gesture recognition,” *Sensors*, vol. 19, no. 12, 2019.

-
- [46] DFRobot, “Gravity: Analog emg sensor by oymotion.” [en línea] <<https://www.dfrobot.com/product-1661.html>> [consulta: 9 junio 2024].
- [47] M. Electronics, “Kit de sensor muscular.” [en línea] <<https://mcielectronics.cl/shop/product/kit-de-sensor-muscular-25509/>> [consulta: 9 junio 2024].
- [48] Mindrove, “armband 8 channel.” [en línea] <https://mindrove.com/product/armband_8_ch/> [consulta: 9 junio 2024].
- [49] H. Hinrichs, M. Scholz, and A. K. e. a. Baum, “Comparison between a wireless dry electrode eeg system with a conventional wired wet electrode eeg system for clinical applications,” *Scientific Reports*, vol. 10, p. 5218, 2020.
- [50] V. Alvarado Castillo, J. Sánchez Flores, J. C. Gómez, E. Chihuan Huayta, and C. De La Cruz Casaño, “Adquisición de señales semg con electrodos secos para el control de movimiento de dedos en una prótesis robótica fabricada en una impresora 3d,” *Ingeniare. Revista chilena de ingeniería*, vol. 27, no. 3, pp. 522–536, 2019.
- [51] Myoware, “Conductive fabric electrodes.” [en línea] <<https://myoware.com/project/conductive-fabric-electrodes/>> [consulta: 9 junio 2024].
- [52] SQLite Development Team, “Sqlite.” [en línea] <<https://www.sqlite.org/>> [consulta: 19 octubre 2024].
- [53] M. D. Olmo and R. Domingo, “Emg characterization and processing in production engineering,” *Materials (Basel, Switzerland)*, vol. 13, no. 24, p. 5815, 2020.
- [54] K. H. Lee, J. Y. Min, and S. Byun, “Electromyogram-based classification of hand and finger gestures using artificial neural networks,” *Sensors*, vol. 22, no. 1, p. 225, 2021.

-
- [55] A. Sultana, M. T. I. Opu, F. Ahmed, and M. S. Alam, “A novel machine learning algorithm for finger movement classification from surface electromyogram signals using welch power estimation,” *Healthcare Analytics*, vol. 5, p. 100296, 2024.
- [56] E. Farago, D. MacIsaac, M. Suk, and A. D. C. Chan, “A review of techniques for surface electromyography signal quality analysis,” *IEEE Reviews in Biomedical Engineering*, vol. 16, pp. 472–486, 2023.
- [57] DFRobot, “Sku:sen0240.” [en línea] <https://wiki.dfrobot.com/Analog_EMG_Sensor_by_OYMotion_SKU_SEN0240> [consulta: 8 agosto 2024].

ANEXO

A. DIMENSIONES DE LOS SENSORES

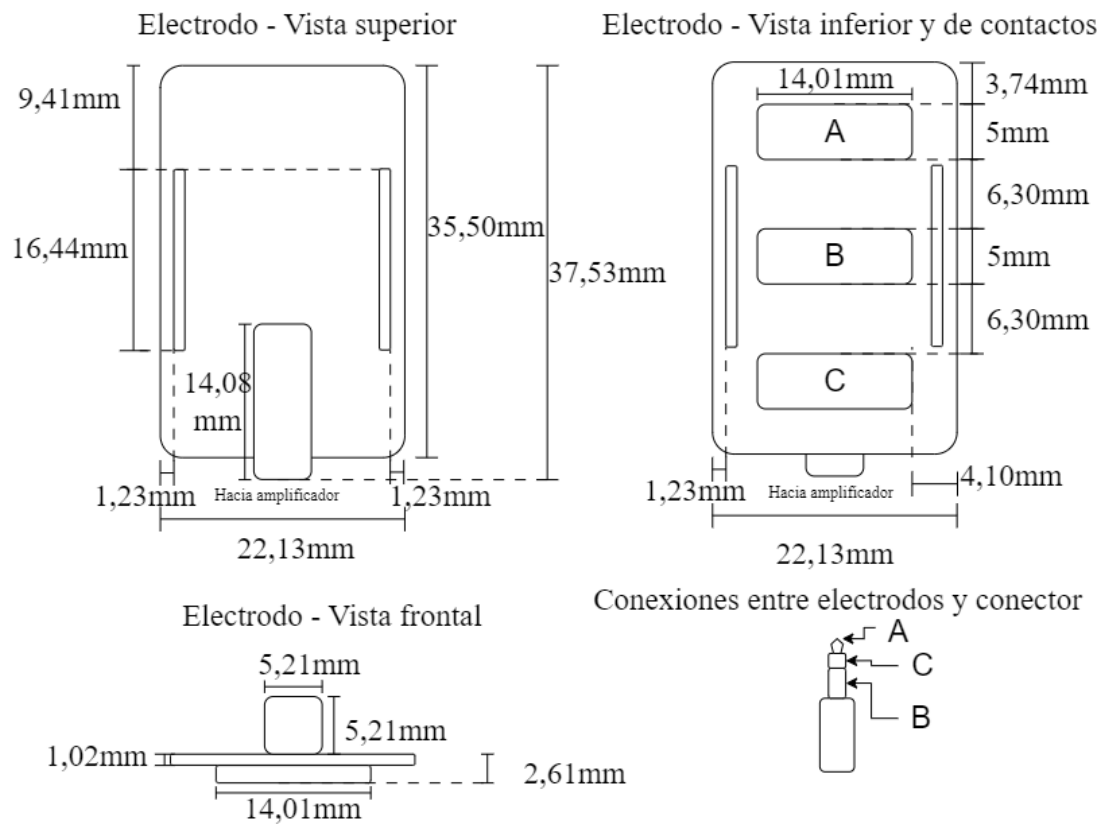


Fig. A.1: Dimensiones de las placas de electrodos

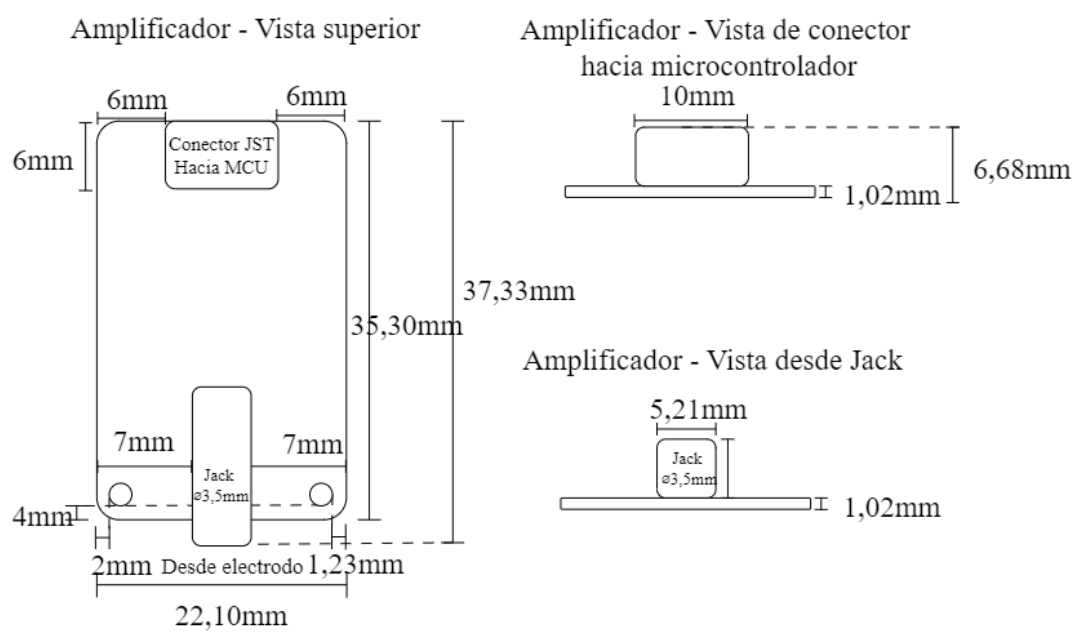


Fig. A.2: Dimensiones de las placas de amplificadores

B. CÓDIGO EN ARDUINO PARA RETORNAR LAS LECTURAS ANÁLOGAS

```
/* Retornar lectura de ADC de 3 canales + función de Onset
Sketch de Arduino para retornar por consola serial los valores recibidos desde los ADCs
.
Retorna valores según lo recibido en cada pin analógico
Incluye una función para mantener pulsado un botón y mandar su estado por serial
*/

#define SAMPLE_FREQ 1000 // Frecuencia de muestreo en Hz
#define SAMPLE_PERIOD_US (1000000 / SAMPLE_FREQ) // Periodo de muestreo en
microsegundos

// Definición de pines
#define CH1_PIN A0
#define CH2_PIN A1
#define CH3_PIN A2
#define ONSET_PIN A3
unsigned long lastSampleTime = 0; // Variable para almacenar el tiempo del último muestreo

// Configuración inicial
void setup() {
    // Inicializar la comunicación serial a 115200 baud
    Serial.begin(115200);

    // Configurar los pines A0, A1, A2 y A3 como entradas
    pinMode(CH1_PIN, INPUT); // Canal 1
    pinMode(CH2_PIN, INPUT); // Canal 2
    pinMode(CH3_PIN, INPUT); // Canal 3
}
```

```

    pinMode(ONSET_PIN, INPUT); // Onset (1 si se mantiene pulsado el botón)
}

// Bucle principal
void loop() {
    // Obtener el tiempo actual en microsegundos
    unsigned long currentTime = micros();

    // Verificar si ha pasado el periodo de muestreo
    if (currentTime - lastSampleTime >= SAMPLE_PERIOD_US) {
        // Actualizar el tiempo del último muestreo
        lastSampleTime = currentTime;

        // Leer los valores analógicos desde los pines A0, CH2_PIN y A2
        int sensorValue1 = analogRead(CH1_PIN); // Canal 1
        int sensorValue2 = analogRead(CH2_PIN); // Canal 2
        int sensorValue3 = analogRead(CH3_PIN); // Canal 3

        // Leer estado del botón de onset
        // Mantenerlo presionado mientras se haga un gesto
        int onset = digitalRead(ONSET_PIN);

        // Enviar los valores de voltaje a través de la consola serial
        // Formato: <onset>,<CH1>,<CH2>,<CH3>
        Serial . print (onset);
        Serial . print ("");
        Serial . print (sensorValue1);
        Serial . print ("");
        Serial . print (sensorValue2);
        Serial . print ("");
        Serial . println (sensorValue3);
    }
}

```

C. SCRIPTS EN PYTHON

C.1. *Lectura de 3 canales y onset*

Nombre del archivo: `lectura_3ch_rawEMG.py`

''' Lectura de 3 canales EMG bruto

Script en Python para registrar los datos de EMG recibidos desde el puerto
tserial

Espera recibirlos con el formato <onset>,<CH1>,<CH2>,<CH3>,...

Estructura de la base de datos

id: Identificador único autoincremental para cada entrada

gesto_id: Identificador único para cada gesto. Útil para diferenciar
distintas instancias del mismo gesto

sesion_id: Número de la sesión en que se registró el gesto

onset: Indicador de si se está ejecutando el gesto. 1 para indicar que
está en ejecución

nombre_gesto: Nombre del gesto hecho

fs: Frecuencia de muestreo en Hertz

fecha: Fecha en la que se hizo la captura, YYYY-MM-DD HH:MM:SS

CHX: Valor recibido para el canal X sin pasar por filtros

Uso

- Ejecutar e ingresar el número de la sesión actual, o pulsar Enter para mantener el último usado
- Mantener pulsado botón en la placa mientras se hace el gesto

-
- Hacer 1 repetición de 1 gesto por vez
 - Al terminar la toma de datos pulsar Ctrl+C

Bastían Rivas

'''

```
import serial
import sqlite3
import time
from datetime import datetime
import os

# Función para insertar datos en la base de datos en lotes
def insertar_datos_lote (datos):
    cursor.executemany("""INSERT INTO datos (gesto_id, sesion_id , nombre_gesto,
                                onset , CH1, CH2, CH3, fecha, fs)
                                VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)""", datos)

    conexion.commit()

# Configurar el puerto serial
puerto_serial = 'COM4'
baud_rate = 115200

# Frecuencia de muestreo en Hz
fs = 1000

# Conectar o crear la base de datos SQLite
db_path = 'Datos/ datos_gestos_3ch .db'
nombre_tabla = 'raw'
conexion = sqlite3 .connect(db_path)
cursor = conexion.cursor ()

# Mostrar la ubicación de la base de datos en consola
print (f"Usando base de datos en: {os.path.abspath(db_path)}")
```

Crear la tabla si no existe

```
cursor.execute( ''' CREATE TABLE IF NOT EXISTS datos (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    gesto_id INTEGER,
                    sesion_id INTEGER,
                    onset INTEGER,
                    nombre_gesto TEXT,
                    fs INTEGER,
                    fecha TEXT,
                    CH1 INTEGER,
                    CH2 INTEGER,
                    CH3 INTEGER
                ) '' )
```

Obtener el último gesto_id y sesion_id registrados en la base de datos

```
cursor.execute( f''''SELECT MAX(gesto_id), nombre_gesto, sesion_id
                    FROM {nombre_tabla}'''' )

ultimo_registro = cursor.fetchone()
ultimo_gesto_id = ultimo_registro [0] if ultimo_registro [0] is not None else 0
if ultimo_registro [1] is not None:
    ultimo_nombre_gesto = ultimo_registro [1]
else:
    ultimo_nombre_gesto = "N/A"

ultimo_sesion_id = ultimo_registro [2] if ultimo_registro [2] is not None else 1
gesto_id = ( ultimo_gesto_id + 1) if ultimo_gesto_id is not None else 1

# Solicitar el sesion_id al usuario
print( f"Última sesión registrada con ID = { ultimo_sesion_id }")
sesion_id = input( "Por favor, ingrese el número de sesión: ")
sesion_id = int( sesion_id ) if sesion_id else ultimo_sesion_id
```

```

# Solicitar el nombre del gesto al usuario
print(f"Último gesto registrado fue '{ultimo_nombre_gesto}' con
ID = { ultimo_gesto_id }")
nombre_gesto = input("Por favor, ingrese el nombre del gesto: ")
fecha = datetime.now().strftime(' %Y-%m-%d %H:%M:%S')

# Lista para almacenar los datos leídos
# Uso un lote acá porque escribir directamente en la base de datos es demasiado
# lento
datos_lote = []

# Abrir el puerto serial y comenzar a leer datos
try:
    with serial.Serial( puerto_serial , baud_rate , timeout=1) as ser:
        print(f"Leyendo desde { puerto_serial } a { baud_rate} baud ...
        \nFinalizar con Ctrl+C")
    while True:
        # Leer línea desde el puerto serial
        if ser.in_waiting > 0:
            try:
                data = ser.readline().decode('ascii').rstrip()
                valores = data.split(',')

                # Asegurarse de que hay cuatro valores
                # Formato: onset, ch1, ch2, ch3
                if len(valores) == 4:
                    try:
                        onset = int(valores[0])
                        ch1 = int(valores[1])
                        ch2 = int(valores[2])
                        ch3 = int(valores[3])
                        datos_lote.append(( gesto_id , sesion_id ,
                                         nombre_gesto, onset, ch1, ch2,
                                         ch3, fecha, fs))

```

```

        print(f"Registrado Onset: {onset}\t CH1: {ch1}\t
              CH2: {ch2}\t CH3: {ch3}")

        # Insertar en la base de datos en lotes de
        # 100 registros
        if len( datos_lote ) >= 100:
            insertar_datos_lote ( datos_lote )
            # Limpiar la lista después de insertar
            datos_lote = []
    except ValueError:
        print(f"Error al convertir los datos a enteros :
              {data}")
    else :
        print(f"Datos incompletos recibidos : {data}")

except UnicodeDecodeError:
    # Suele caer acá cuando registra datos incompletos desde el
    # puerto serial . Típicamente pasa si empieza a leer cuando
    # se está recibiendo una línea
    pass

except serial . SerialException as e:
    print(f"Error al acceder al puerto serial : {e}")
except KeyboardInterrupt :
    print(f"\nLectura interrumpida . Datos guardados en {db_path}")
    # Insertar cualquier dato restante en la base de datos
    if datos_lote :
        insertar_datos_lote ( datos_lote )

# Cerrar la conexión a la base de datos
finally :
    conexion.close ()

```

C.2. Consultar los gestos registrados en la base de datos

Nombre del archivo: `consultar_gestos.py`

```
''' Consultar gestos
```

```
Script en Python para consultar qué gestos existen en la base de datos.
```

```
Tras ejecutarlo retorna por consola la ID única, fecha de captura y el nombre de cada gesto registrado .
```

```
Bastían Rivas
```

```
'''
```

```
import sqlite3
```

```
import os
```

```
# Conectar a la base de datos SQLite
```

```
db_path = 'Datos/ datos_gestos_3ch .db'
```

```
nombre_tabla = 'raw'
```

```
conexion = sqlite3 .connect(db_path)
```

```
cursor = conexion.cursor()
```

```
# Mostrar la ubicación de la base de datos en consola
```

```
print(f'Usando base de datos en: {os.path.abspath(db_path)}')
```

```
# Consultar los datos de gesto_id , fecha y gesto , omitiendo gesto_id repetidos
```

```
cursor.execute(f''''''
```

```
    SELECT gesto_id, MIN(fecha), nombre_gesto, sesion_id
```

```
    FROM {nombre_tabla}
```

```
    GROUP BY gesto_id
```

```
''''')
```

```
datos = cursor . fetchall ()
```

```
# Imprimir los datos obtenidos por consola
```

```
print(f''''''--- Gestos registrados ---
```

```
ID \tFecha \tSesión\tNombre del gesto
```

```
----\t-----\t-----\t-----
```

```
""")
for fila in datos:
    gesto_id, fecha, gesto, sesion_id = fila
    print(f"{gesto_id}\t{fecha}\t{sesion_id}\t{gesto}")

# Cerrar la conexión a la base de datos
conexion.close()
```

C.3. Graficar los 3 canales de un gesto en bruto

Nombre del archivo: graficar_datos_3ch.py

```
''' Graficar datos
```

```
Script en Python para visualizar los datos de sensor EMG capturados.  
Al ejecutarlo imprime los gestos disponibles con sus fechas de IDs respectivas .  
Pide la ID única de un gesto para graficarlo
```

```
Bastían Rivas
```

```
'''
```

```
import sqlite3  
import matplotlib.pyplot as plt  
import matplotlib as mpl  
import numpy as np  
import os  
  
# Conectar a la base de datos SQLite  
db_path = 'Datos/ datos_gestos_3ch .db'  
nombre_tabla = 'raw'  
conexion = sqlite3 .connect(db_path)  
reescalado = 5.0/1023  
# Mostrar la ubicación de la base de datos en consola  
print(f"Usando base de datos en {os.path.abspath(db_path)}\n")  
cursor = conexion.cursor()  
  
# Consultar los datos de gesto_id , fecha y gesto , omitiendo gesto_id repetidos  
cursor.execute(f"""  
    SELECT gesto_id, MIN(fecha), sesion_id , nombre_gesto  
    FROM {nombre_tabla}  
    GROUP BY gesto_id  
""")
```

```

datos = cursor . fetchall ()

# Mostrar todos los gestos registrados en la base de datos
print (f"----- Gestos registrados ----
ID \tFecha \tSesión\tNombre del gesto
-----\t-----\t-----\t-----
")

for fila in datos:
    gesto_id , fecha , sesion_id , nombre_gesto = fila
    print (f"{ gesto_id }\t{ fecha }\t{ sesion_id }\t{ nombre_gesto }")

# Obtener el gesto_id con la mayor ID por defecto
cursor . execute (f"SELECT MAX(gesto_id) FROM {nombre_tabla}")
max_gesto_id = cursor . fetchone () [0]

# Pedir el gesto_id del gesto a graficar
gesto_id_input = input (f"Por favor , introduce la ID del gesto a graficar
(Enter para usar el más nuevo): ")

# Usar el gesto_id ingresado o el mayor ID por defecto
gesto_id = int ( gesto_id_input ) if gesto_id_input else max_gesto_id

#for gesto_id in range(106,124):
if gesto_id :

    # Obtener los datos del gesto
    cursor . execute (f"SELECT fs, onset, CH1 * {reescalado}, CH2 * {reescalado},
CH3 * {reescalado}, nombre_gesto
FROM {nombre_tabla}
WHERE gesto_id = ?",
( gesto_id ,))

```

```

datos = cursor . fetchall ()
nombre_gesto = datos [0][ -1] # Obtener el nombre del gesto

# Separar los datos en listas para graficar y análisis
fs = datos [0][0] # La frecuencia de muestreo es igual para todos los datos
onset_values = [dato [1] for dato in datos]

# Lista de valores por canal
CH1_values = [dato [2] for dato in datos]
CH2_values = [dato [3] for dato in datos]
CH3_values = [dato [4] for dato in datos]

# Generar el vector de tiempo
N = len(CH1_values)
T = 1.0 / fs # Periodo de muestreo en segundos
time_vector = np.linspace (0.0, N*T, N)

# Encontrar los puntos de cambio en el onset
onset_changes = [i for i in range(1, len(onset_values))
                 if onset_values[i] != onset_values[i-1]]

# Tamaños de fuente
titulo_size = 12
tick_size = 8
label_size = 10

# Graficar los datos
mpl.rc('font', family='Times New Roman')
fig, axs = plt.subplots(3, 1, figsize=(2, 2.5))
fig.suptitle(f'Señal bruta de\n{nombre_gesto}', fontsize = titulo_size)
plt.subplots_adjust(left = 0.25, right=0.95, bottom=0.19, top = 0.85)

```

```

# Límites eje Y
yinf = -0.3
ysup = 3.3

# Graficar CH1
axs [0]. plot ( time_vector , CH1_values, label='CH1')
axs [0]. set_ylabel ('CH1[V]', fontsize = label_size )
axs [0]. set_ylim (yinf, ysup)
axs [0]. tick_params (labelbottom=False) # Omite los números del eje X
axs [0]. grid ()
for change in onset_changes :
    axs [0]. axvline (x=time_vector [change], color='red', linestyle ='—')

# Graficar CH2
axs [1]. plot ( time_vector , CH2_values, label='CH2')
axs [1]. set_ylabel ('CH2[V]', fontsize = label_size )
axs [1]. set_ylim (yinf, ysup)
axs [1]. tick_params (labelbottom=False) # Omite los números del eje X
axs [1]. grid ()
for change in onset_changes :
    axs [1]. axvline (x=time_vector [change], color='red', linestyle ='—')

# Graficar CH3
axs [2]. plot ( time_vector , CH3_values, label='CH3')
axs [2]. set_ylabel ('CH3[V]', fontsize = label_size )
axs [2]. set_xlabel ('Tiempo [s]', fontsize = label_size )
axs [2]. set_ylim (yinf, ysup)
axs [2]. grid ()
for change in onset_changes :
    axs [2]. axvline (x=time_vector [change], color='red', linestyle ='—')

# Ajustar el layout
plt . tight_layout ()

```

```
# Guardar gráfico generado
directorio = 'fig_3ch/raw'
if not os.path.exists ( directorio ):
    os.makedirs( directorio )
nombre_archivo = f"{ gesto_id }-{ nombre_gesto }.png"
ruta_archivo = os.path.join( directorio , nombre_archivo)
plt . savefig ( ruta_archivo )
print (f"Guardando gráfico en { ruta_archivo }")

# Mostrar la gráfica
#plt.show()

# Cerrar la conexión a la base de datos
conexion.close ()
```

C.4. Normalizar los gestos

Nombre del archivo: `emg_cvm_norm_sql.py`. Todos los créditos correspondientes van a los autores originales en [42]

```
# -*- coding: utf-8 -*-
```

```
"""Ajustando una señal electromiográfica funcional:
```

```
-Laboratorio Integrativo de Biomecánica y Fisiología del Esfuerzo,  
Escuela de Kinesiología, Universidad de los Andes, Chile-  
-Escuela de Ingeniería Biomédica, Universidad de Valparaíso, Chile-  
--Profesores: Oscar Valencia & Alejandro Weinstein--
```

Modificado por Bastián Rivas para ser usado en el trabajo "Caracterización de sensores EMG en gestos de mano"

Incluye una serie de funciones nuevas para trabajar con una base de datos en SQLite y algunos ajustes para seguir con la línea del trabajo

Todos los créditos van a sus respectivos autores

Changelog:

18-12:

- Modificado el ejemplo para usar SQLite en lugar de csv
- Reescalado el gráfico generado para facilitar su lectura desde un documento impreso
- Agregada interacción con el script por medio de consola

20-12:

- Calcular SNR de una señal
- Calcular FFT de una señal
- Generar una nueva ventana para mostrar lo nuevo

21-12:

- Automatizada la elección de CVM y ruido en base a sesión

22-12:

- Automatizado todo el procesamiento por cada gesto y canal
- Movidos los cálculos de fft a 'generar_tabla_fft_gestos .py' para mantener modularidad

24-12:

- Agregado un ejemplo con el que se grafica a partir de la base de datos

"""

Importar librerías

import numpy as np

from scipy.signal import butter, filtfilt

import matplotlib.pyplot as plt

Nuevo: para trabajar con sqlite

import sqlite3

import os

Nuevo: para cambiar el tipo de fuente de los gráficos

import matplotlib as mpl

def ajusta_emg_func(emg_fun, emg_cvm, fs, fc, forden):

"""Ajusta EMG funcional según contracción voluntaria máxima.

La función utiliza una señal EMG funcional y otra basada en la
solicitud de una contracción isométrica voluntaria máxima. Ambas señales
son procesadas considerando su centralización (eliminación de
"offset"), rectificación y filtrado (pasa bajo con filtfilt).

Parameters

emg_fun : array_like

EMG funcional del músculo a evaluar

emg_cvm : array_like

```

    EMG vinculada a la contracción voluntaria máxima del mismo músculo
fs : float
    Frecuencia de muestreo, en hertz , de la señal EMG. Debe ser la misma
    para ambas señales .
fc : float
    Frecuencia de corte , en hertz , del filtro pasa-bajos.
forden : int
    Orden del filtro pasa bajos

Return
-----
emg_fun_norm : array_like
    EMG funcional filtrada y normalizada
emg_fun_env_f : array_like
    Envolvente de EMG funcional filtrada
emg_cvm_envf_ : array_like
    Envolvente de EMG CVM filtrada
"""
#centralizando y rectificando las señales EMG
emg_fun_env = abs(emg_fun - np.mean(emg_fun))
emg_cvm_env = abs(emg_cvm - np.mean(emg_cvm))

# Filtrado pasa-bajo de las señales
b, a = butter (int (forden), (int (fc)/( fs/2)), btype = 'low')
emg_fun_env_f = filtfilt (b, a, emg_fun_env)
emg_cvm_env_f = filtfilt (b, a, emg_cvm_env)

#calculando el valor máximo de emg_cvm y ajustando la señal EMG funcional
emg_cvm_I = np.max(emg_cvm_env_f)
emg_fun_norm = (emg_fun_env_f / emg_cvm_I) * 100

return emg_fun_norm, emg_fun_env_f, emg_cvm_env_f

```

```

# %%

def plot_emgs(emg_fun, emg_fun_env, emg_fun_norm, emg_cvm, emg_cvm_env,
              fs, f_c, f_orden,
              nombre):
    """Grafica señales de EMG funcional y CVM.

    Parameters
    -----
    emg_fun : array_like
        EMG funcional.
    emg_fun_env : array_like
        Envolvente del EMG funcional.
    emg_fun_norm : array_like
        EMG funcional normalizada según CVM.
    emg_cvm : array_like
        EMG contracción voluntaria máxima.
    fs : float
        Frecuencia de muestreo, en hertz.
    f_c : float
        Frecuencia de corte del filtro pasa-bajo, en hertz.
    f_orden : int
        Orden del filtro.
    nombre : str
        Nombre del gesto.

    Notes
    -----
    Cambiados algunos tamaños y parte del título. Las líneas originales están
    comentadas

    """
    # Tamaños de fuente
    titulo_size = 15

```

```
label_size = 12
```

```
# Vectores de tiempo
```

```
t1 = np.arange(0, len(emg_fun) / fs, 1 / fs)
```

```
t2 = np.arange(0, len(emg_cvm) / fs, 1 / fs)
```

```
#fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize = (8, 7))
```

```
fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize =(4, 5))
```

```
fig.canvas.manager.set_window_title('Normalización con CVM')
```

```
ax1.plot(t1, emg_fun, 'b', label='Señal bruta')
```

```
#ax1.set_title(f'Músculo: {nombre};\nfiltro aplicado: f.c={f.c} [Hz] de
```

```
# 'f'orden {f.orden}')
```

```
ax1.set_title(f'Gesto: {nombre};\nFiltro aplicado: f.c={f.c} [Hz] de\norden  
{f.orden}', fontsize = titulo_size)
```

```
ax1.plot(t1, emg_fun_env, 'r', lw=2, label='Señal filtrada')
```

```
ax1.set_ylabel(f'{nombre} Funcional\nAmplitud [V]', fontsize = label_size)
```

```
#ax1.set_ylim(emg_fun.min() - 0.1, emg_fun.max() + 0.1)
```

```
ax1.set_ylim(0, emg_fun.max() + 0.1)
```

```
ax1.set_xlim(0, t1.max())
```

```
ax1.grid()
```

```
ax1.legend(loc='upper center', fontsize='x-small', borderpad=None)
```

```
ax2.plot(t2, emg_cvm, 'b', label='Señal bruta')
```

```
ax2.plot(t2, emg_cvm_env, 'r', lw=2, label='Señal filtrada')
```

```
ax2.set_ylabel(f'{nombre} CVM\nAmplitud [V]', fontsize=label_size)
```

```
ax2.axvline((np.argmax(emg_cvm_env) / fs), color='maroon')
```

```
ax2.text(0.85, 0.95, f'Max = {emg_cvm_env.max():.2f}',  
        transform=ax2.transAxes, ha="left", va="top")
```

```
#ax2.set_ylim(emg_cvm.min() - 0.1, emg_cvm.max() + 0.1)
```

```
#ax2.set_ylim(0, emg_cvm.max() + 0.1)
```

```
ax2.set_xlim(0, t2.max())
```

```
ax2.grid()
```

```

ax2.legend(loc='upper center', fontsize='x-small', borderpad=None)

ax3.plot(t1, emg_fun_norm, 'g', label='Señal ajustada según CVM')
ax3.set_ylim(emg_fun_norm.min(), emg_fun_norm.max() + 2)
ax3.set_xlim(0, t1.max())
ax3.set_xlabel('Tiempo [s]', fontsize=label_size)
ax3.set_ylabel('% EMG CVM')
ax3.grid()
ax3.legend(loc='upper center', fontsize='x-small', borderpad=None)

plt.tight_layout(h_pad=.1)

```

```

# %% Nuevo: Función para calcular RMS

```

```

def get_rms(values):

```

```

    """

```

```

    Calcula el valor RMS de una lista de valores .

```

```

    Parameters

```

```

    -----

```

```

    values ( list or np.array): Lista de valores numéricos.

```

```

    Returns:

```

```

        float : El valor RMS de la lista .

```

```

    """

```

```

    values = np.array(values)

```

```

    rms = np.sqrt(np.mean(np.square(values)))

```

```

    return rms

```

```

# %% Nuevo: Función para calcular la SNR

```

```

import numpy as np

```

```

def get_SNR(signal_rms, noise_rms):

```

```

    """

```

Calcula la relación señal a ruido (SNR) a partir de los valores RMS de la señal y del ruido.

Parameters

signal_rms (float): El valor RMS de la señal.

noise_rms (float): El valor RMS del ruido.

Returns:

float : El valor de la SNR en decibeles (dB).

"""

```
snr = 20 * np.log10(signal_rms / noise_rms)
```

```
return snr
```

###

```
def normalizar_3ch_sql ( gesto_id , ruta_db = 'Datos/ datos_gestos_3ch .db' ,  
                        tabla_raw = 'raw' , fs = 1000, fc = 150, forden = 2,  
                        reescalado = 5.0/1023, canales = [1, 2, 3]):  
    """
```

Script para normalizar señales de hasta tres canales de un gesto específico a partir de su ID, almacenado en una base de datos en SQLite.

Requiere que en la base de datos existan hasta 3 canales de registros del gesto a usar, un registro llamado 'Reposo' y otros llamados 'CVM CH1', 'CVM CH2' y 'CVM CH3, todos correspondiendo a la misma sesión en la que se tomaron los registros

Parameters

”gesto_id”: Número identificador del gesto a normalizar

”ruta_db”: La ruta a la base de datos SQLite que por defecto es
'Datos/ datos_gestos_3ch .db'.

”tabla_raw”: El nombre de la tabla dentro de la base de datos donde se registran los datos brutos

”fs”: La frecuencia de muestreo, cuyo valor por defecto es 1000 Hz.

”fc”: La frecuencia de corte para el filtrado , cuyo valor por defecto es 150 Hz.

”forden”: El orden del filtro pasabajos , cuyo valor por defecto es 2.

”reescalado”: Factor para reescalar los datos, cuyo valor por defecto es de 5.0/1023 para una tarjeta que recibe hasta 5 volts en un ADC de 10 bits

”canales”: Lista de canales a analizar , que por defecto es [1, 2, 3]

Este script realiza las siguientes operaciones :

1. Conectar a la base de datos SQLite especificada .
2. Obtener los datos de la señal para el ”gesto_id” proporcionado .
3. Filtrar las señales con la frecuencia y el orden especificado
4. Calcular las envolventes de los 3 canales
5. Normalizar las señales a partir de su contracción voluntaria máxima (CVM)

Return

datos_normalizados : array_like

Un diccionario con las siguientes entradas :

id (int)	: Identificador único autoincremental para cada entrada
gesto_id (int)	: Identificador único para cada gesto . Útil para diferenciar distintas instancias del mismo gesto
sesion_id (int)	: ID de la sesión en la que se hizo la captura
onset (list , int)	: Indicador de si se está ejecutando el gesto . 1 para indicar que está en ejecución

```

nombre_gesto ( string )      : Nombre del gesto hecho
fs ( int )                   : Frecuencia de muestreo en Hertz
fc ( int )                   : Frecuencia de corte del filtro
fecha ( string )             : Fecha en la que se hizo la captura ,
                               -MM-DD HH:MM:SS

ch1_env_fil ( list , float ) : Valores de la envolvente post
                               filtrado del canal 1
ch1_norm ( list , float )    : Valores del canal 1 normalizado
                               respecto a la CVM correspondiente
ch2_env_fil ( list , float ) : Valores de la envolvente post
                               filtrado del canal 2
ch2_norm ( list , float )    : Valores del canal 2 normalizado
                               respecto a la CVM correspondiente
ch3_env_fil ( list , float ) : Valores de la envolvente post
                               filtrado del canal 3
ch3_norm ( list , float )    : Valores del canal 3 normalizado
                               respecto a la CVM correspondiente

"""

# Conectarse a la base de datos
conexion = sqlite3 .connect(ruta_db)
cursor = conexion.cursor()

# Obtener datos del gesto funcional
cursor.execute(f"""
    SELECT fs, fecha, onset, sesion_id , CH1, CH2, CH3, nombre_gesto
    FROM {tabla_raw}
    WHERE gesto_id = ?
""", ( gesto_id ,))

datos_gesto_funcional = cursor . fetchall ()

# Nuevo: ahora lee los 3 canales sin tener que ingresarlos manualmente

```

```

canales = [1, 2, 3]

# Diccionario para ir registrando los datos que corresponden a cada canal
df_funcional = {
    1: [],
    2: [],
    3: []
}

for num_canal in canales:
    # df_funcional[i] corresponde al canal i
    df_funcional[num_canal] = [dato[3 + num_canal] * reescalado
                               for dato in datos_gesto_funcional ]
    # Formato: [fs, fecha, onset, sesion_id, CH1, CH2, CH3, nombre_gesto ]

# Clonar vector de onset del gesto. Este vector aplica para los 3 canales
onset_funcional = [dato[2] for dato in datos_gesto_funcional ]

# Estos datos son los mismos para los 3 canales a través de todo el tiempo
fecha_captura = datos_gesto_funcional [0][1]
sesion_id_gesto = datos_gesto_funcional [0][3]
nombre_gesto_funcional = datos_gesto_funcional [0][-1]

# Filtrar datos de CVM según el canal en uso y el número de sesión
# Nuevo: ahora lee los 3 canales en lugar de solo uno
datos_cvm = {1: [], 2: [], 3: []}
df_cvm = {1: [], 2: [], 3: []}
emg_funcional = {1: [], 2: [], 3: []}
emg_cvm = {1: [], 2: [], 3: []}
emg_f_n = {1: [], 2: [], 3: []}
emg_f_env = {1: [], 2: [], 3: []}
emg_cvm_env = {1: [], 2: [], 3: []}

for num_canal in canales:
    canal_string = "CH" + str(num_canal)
    # Este query retorna una lista con la ID del CVM, la fecha, su nombre,

```

```
# la sesión en la que se grabó y los valores del canal correspondiente
```

```
cursor.execute(f"""
```

```
    SELECT gesto_id, fs, fecha, onset, sesion_id, { canal_string },
```

```
    nombre_gesto
```

```
    FROM {tabla_raw}
```

```
    WHERE nombre_gesto LIKE ' %CVM {canal_string} %'
```

```
    AND sesion_id = ?
```

```
""", ( sesion_id_gesto , ) )
```

```
datos_cvm[num_canal] = cursor.fetchall ()
```

```
df_cvm[num_canal] = [dato[-2] * reescalado
```

```
    for dato in datos_cvm[num_canal]]
```

```
emg_funcional[num_canal] = np.array ( df_funcional [num_canal])
```

```
emg_cvm[num_canal] = np.array(df_cvm[num_canal])
```

```
# Línea para invocar la función de ajuste
```

```
emg_f_n[num_canal],
```

```
emg_f_env[num_canal],
```

```
emg_cvm_env[num_canal] = ajusta_emg_func(emg_funcional[num_canal],
```

```
    emg_cvm[num_canal], fs, fc,
```

```
    forden)
```

```
# emg_f_n: Señal emg normalizada respecto a la CVM
```

```
# emg_f_env: Envoltente de la señal luego de ser filtrada
```

```
# emg_cvm_env: Envoltente de la señal CVM normalizada y filtrada
```

```
# Valores a retornar :
```

```
resultado_normalizado = {
```

```
    ' gesto_id ': gesto_id ,
```

```
    ' sesion_id ': sesion_id_gesto ,
```

```
    ' fecha ': fecha_captura ,
```

```
    ' nombre_gesto ': nombre_gesto_funcional ,
```

```

    'fs': fs,
    'fc': fc,
    'onset': onset_funcional,
    # Canal 1
    'ch1_env_fil' : emg_f_env[1],# Envolvente después de filtrar señal EMG
    'ch1_norm'    : emg_f_n[1], # Señal EMG normalizada respecto a la CVM
    # Canal 2
    'ch2_env_fil' : emg_f_env[2],# Envolvente después de filtrar señal EMG
    'ch2_norm'    : emg_f_n[2], # Señal EMG normalizada respecto a la CVM
    # Canal 3
    'ch3_env_fil' : emg_f_env[3],# Envolvente después de filtrar señal EMG
    'ch3_norm'    : emg_f_n[3], # Señal EMG normalizada respecto a la CVM
}

return resultado_normalizado

#%%

def registrar_datos_norm (datos_normalizados,
                        ruta_db='Datos/ datos_gestos_3ch .db',
                        tabla_norm = 'norm'):
    """
    Registra los datos normalizados en la base de datos.

    Estructura de la base de datos
    -----
    id: Identificador único autoincremental para cada entrada
    gesto_id: Identificador único para cada gesto. Útil para diferenciar
            distintas instancias del mismo gesto
    sesion_id: ID de la sesión en la que se hizo la captura
    fecha: Fecha en la que se hizo la captura, YYYY-MM-DD HH:MM:SS
    nombre_gesto: Nombre del gesto hecho
    fs: Frecuencia de muestreo en Hertz ( default : 1000)
    fc: Frecuencia de corte del filtro de 2do grado ( default : 150)

```

```

        onset: Indicador de si se está ejecutando el gesto. 1 para indicar que
                está en ejecución
        chX_norm: Valor del canal X normalizado respecto a la CVM correspondiente
        chX_env_fil: Valor de la envolvente post filtrado del canal X

"""

conexion = sqlite3 .connect(ruta_db)
cursor = conexion.cursor()

# Crear la tabla "norm" y ejecutar la consulta para crearla

cursor .execute(f"""
CREATE TABLE IF NOT EXISTS {tabla_norm} (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    gesto_id INTEGER,
    sesion_id INTEGER,
    fecha TEXT,
    nombre_gesto TEXT,
    fs INTEGER,
    fc INTEGER,
    onset INTEGER,
    ch1_norm REAL,
    ch1_env_fil REAL,
    ch2_norm REAL,
    ch2_env_fil REAL,
    ch3_norm REAL,
    ch3_env_fil REAL
);
""")

# Descomponer y registrar los valores de cada canal
n_registros = len(datos_normalizados['onset'])

for i in range( n_registros ):

```

```

insertar_query = f"""
INSERT INTO {tabla_norm} (gesto_id, sesion_id , fecha, nombre_gesto, fs ,
                        fc, onset, ch1_env_fil , ch1_norm,
                        ch2_env_fil , ch2_norm, ch3_env_fil , ch3_norm)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
"""

valores = (
    datos_normalizados [ ' gesto_id ' ],
    datos_normalizados [ ' sesion_id ' ],
    datos_normalizados [ ' fecha ' ],
    datos_normalizados [ ' nombre_gesto ' ],
    datos_normalizados [ ' fs ' ],
    datos_normalizados [ ' fc ' ],
    datos_normalizados [ ' onset ' ][ i ],
    datos_normalizados [ ' ch1_env_fil ' ][ i ],
    datos_normalizados [ ' ch1_norm ' ][ i ],
    datos_normalizados [ ' ch2_env_fil ' ][ i ],
    datos_normalizados [ ' ch2_norm ' ][ i ],
    datos_normalizados [ ' ch3_env_fil ' ][ i ],
    datos_normalizados [ ' ch3_norm ' ][ i ],

)

cursor.execute( insertar_query , valores )

conexion.commit()
conexion.close ()

print (f"Registrado '{datos_normalizados [ ' nombre_gesto ' ]}' con ID
      {datos_normalizados [ ' gesto_id ' ]} en la tabla '{tabla_norm}'")

# %%

```

```

'''
    Ejemplo de uso: Normalizar todos los registros en la base de datos
'''

# if __name__ == '__main__':
def norm_db_sql():
    # Parámetros de frecuencia y orden de filtro
    tabla_raw = "raw"
    fs = 1e3    # Frecuencia de muestreo
    fc, forden = 150, 2 # Frecuencia de corte y orden del filtro pasabajos

    # Desplegar los gestos disponibles
    ruta_db = 'Datos/ datos_gestos_3ch .db'
    # Factor de reescalado en caso de capturar con un ADC de otra resolución
    # 5.0/1023 es para un ADC de 10 bits que recibe hasta 5 volts
    reescalado = (5.0/1023)
    conexion = sqlite3 .connect(ruta_db)
    cursor = conexion.cursor()

    # Mostrar la ubicación de la base de datos
    print(f"Usando base de datos en: {os.path.abspath(ruta_db)}")

    # Inicio de interacción con usuario
    # Mostrar los gestos almacenados
    cursor.execute(f"""
        SELECT gesto_id, MIN(fecha), nombre_gesto, sesion_id
        FROM {tabla_raw}
        GROUP BY gesto_id
    """)
    datos = cursor . fetchall ()

    # Imprimir los datos obtenidos por consola
    print(f"""--- Gestos registrados ---
ID \tFecha                \tSesión\tNombre del gesto
-----\t-----\t-----\t-----""")

```

```

gestos_a_registrar = []
for fila in datos:
    gesto_id, fecha, gesto, sesion_id = fila
    print(f"{gesto_id}\t{fecha}\t{sesion_id}\t{gesto}")
    gestos_a_registrar.append(gesto_id)

n_gestos = len( gestos_a_registrar )
rpta = []
while rpta not in ["y", "n"]:
    rpta = input(f"¿Normalizar los {n_gestos} gestos? [Y/n]: ").lower()

# Fin de interacción con el usuario
if rpta == "n":
    print("Cancelando ...")
    quit()
else:
    for gesto in gestos_a_registrar:
        # Normalizar todos los gestos
        datos_norm = normalizar_3ch_sql(gesto, ruta_db, tabla_raw, fs, fc,
                                       forden, reescalado, [1,2,3])
        registrar_datos_norm(datos_norm, ruta_db, 'norm')

    print(f"Finalizado . {n_gestos} gestos registrados .")

# Cerrar la conexión a la base de datos
conexion.close()

# %%
if __name__ == '__main__':
    """
    Ejemplo para graficar usando las funciones propuestas y la base de datos.
    Se entrega una cierta interacción con el usuario al ofrecer la lista de

```

```

gestos y consultar por datos como ID de interés y canal a graficar .
También sirve como un ejemplo de cómo hacerle una query a la base de datos y
cómo reescalar los datos brutos
'''

### Consultar gestos
# Conectar a la base de datos SQLite
db_path = 'Datos/ datos_gestos_3ch .db'
conexion = sqlite3 .connect(db_path)
tabla_raw = 'raw'
tabla_norm = 'norm'
cursor = conexion.cursor ()
reescalado = 5.0/1023
fs = 1000 # Frecuencia de sampling en Hz
f_c = 150 # Frecuencia de corte del filtro pasabajos en Hz
f_orden = 2 # Orden del filtro pasabajos en Hz

# Mostrar la ubicación de la base de datos en consola
print (f"Usando base de datos en: {os.path.abspath(db_path)}")

# Consultar los datos de gesto_id , fecha y gesto, omitiendo los gesto_id
# repetidos
cursor .execute(f"""
    SELECT gesto_id, MIN(fecha), nombre_gesto, sesion_id
    FROM {tabla_raw}
    GROUP BY gesto_id
""")
datos = cursor . fetchall ()

# Imprimir los datos obtenidos por consola
print (f"""--- Gestos registrados ---
ID \tFecha \tSesión\tNombre del gesto
-----\t-----\t-----\t-----
""")
for fila in datos:

```

```

gesto_id , fecha , gesto , sesion_id = fila
print (f"{ gesto_id }\t{ fecha }\t{ sesion_id }\t{ gesto }")

gesto_id = int(input("Por favor, introduce la ID del gesto a graficar : "))

# Escoger el canal a graficar
nro_canal = []
canales = [1, 2, 3]
int(input("Por favor, introduce el canal a analizar [1, 2 o 3]: "))

### Obtener datos del gesto funcional bruto
cursor.execute(f"""
    SELECT CH1 * {reescalado}, CH2 * {reescalado}, CH3 * {reescalado},
    nombre_gesto, sesion_id
    FROM {tabla_raw}
    WHERE gesto_id = ?
    """, ( gesto_id ,))

# datos_db es una variable auxiliar donde dejo lo que se retorna del query
datos_db = cursor.fetchall ()

# Nombre para el despliegue
nombre = datos_db [0][−2]

# ID de sesión para obtener la CVM correspondiente
sesion_id = datos_db [0][−1]

# Registrar los datos brutos a usar
emg_fun_list = [dato[nro_canal − 1] for dato in datos_db]
emg_fun = np.array ( emg_fun_list )

### Obtener la envolvente de la señal

```

```

cursor . execute ( f"""
    SELECT ch1_env_fil, ch2_env_fil , ch3_env_fil , sesion_id , nombre_gesto
    FROM {tabla_norm}
    WHERE gesto_id = ?
    """, ( gesto_id , ) )
datos_db = cursor . fetchall ()

```

```

# Registrar los datos de la envolvente del gesto a usar
emg_fun_env_list = [dato[nro_canal - 1] for dato in datos_db]
emg_fun_env = np.array ( emg_fun_env_list )

```

```

### Obtener la señal normalizada
cursor . execute ( f"""
    SELECT ch1_norm, ch2_norm, ch3_norm
    FROM {tabla_norm}
    WHERE gesto_id = ?
    """, ( gesto_id , ) )
datos_db = cursor . fetchall ()

```

```

# Registrar los datos de la señal normalizada a usar
emg_fun_norm_list = [dato[nro_canal - 1] for dato in datos_db]
emg_fun_norm = np.array ( emg_fun_norm_list )

```

```

### Obtener la CVM bruta correspondiente
# Para que quede en formato "CHX" como las columnas de la base de datos
canal_string = str ("CH" + str ( nro_canal ))

```

```

cursor . execute ( f"""
    SELECT {canal_string} * {reescalado}
    FROM {tabla_raw}
    WHERE nombre_gesto LIKE ' %CVM {canal_string} %'
    AND sesion_id = ?

```

```

""" , ( sesion_id , ) )

# Registrar los datos de la CVM bruta
emg_cvm_list = cursor . fetchall ()
emg_cvm = np.array(emg_cvm_list)
#datos_db = cursor . fetchall ()
#emg_cvm = [dato[0] for dato in datos_db[ nro.canal ]]

### Obtener la envolvente de la CVM
cursor . execute ( f """
    SELECT ch1_env_fil, ch2_env_fil , ch3_env_fil
    FROM {tabla_norm}
    WHERE nombre_gesto LIKE ' %CVM {canal_string} %'
    AND sesion_id = ?
    """ , ( sesion_id , ) )

datos_db = cursor . fetchall ()

# Registrar los datos de la envolvente del CVM correcto
emg_cvm_env_list = [dato[ nro.canal - 1] for dato in datos_db]
emg_cvm_env = np.array(emg_cvm_env_list)

# Graficar
mpl.rc( ' font ' , family='Times New Roman')
plot_emgs(emg_fun, emg_fun_env, emg_fun_norm, emg_cvm, emg_cvm_env,
          fs , f_c , f_orden ,
          nombre)

# Guardar gráfico generado
directorio = ' fig_3ch /cvm'
if not os.path . exists ( directorio ):
    os.makedirs( directorio )

```

```
nombre_archivo = f"cvm_{gesto_id}_{nombre}_{canal_string}.png"
ruta_archivo = os.path.join( directorio , nombre_archivo)
plt . savefig ( ruta_archivo )
print (f"Guardando gráfico en { ruta_archivo }")

# Mostrar la gráfica
plt . show()
```

C.5. Obtener el espectro de gestos registrados en bruto

Nombre del archivo: `welch_datos_3ch.py`

```
import sqlite3
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
from scipy.signal import welch
import matplotlib.ticker as mtick
import os

# Conectar a la base de datos SQLite
db_path = 'Datos/ datos_gestos_3ch.db'
nombre_tabla = 'raw'

# Factor de reescalado. En este caso se usa un ADC de 10 bits con 5 volts máx.
reescalado = 5.0/1023

conexion = sqlite3.connect(db_path)
# Mostrar la ubicación de la base de datos en consola
print(f"Usando base de datos en {os.path.abspath(db_path)}\n")
cursor = conexion.cursor()

# Consultar los datos de gesto_id, fecha y gesto, omitiendo gesto_id repetidos
cursor.execute(f"""
    SELECT gesto_id, MIN(fecha), nombre_gesto, sesion_id
    FROM {nombre_tabla}
    GROUP BY gesto_id
""")
datos = cursor.fetchall()

# Imprimir los datos obtenidos por consola
print(f"----- Gestos registrados -----")
print(f"ID \tFecha \tSesión\tNombre del gesto")
print(f"-----\t-----\t-----\t-----")
```

```

"""
)

for fila in datos:
    gesto_id, fecha, gesto, sesion_id = fila
    print(f"{gesto_id}\t{fecha}\t{sesion_id}\t{gesto}")

# Pedir el gesto_id del gesto a graficar
gesto_id = int(input("Por favor, introduce la ID del gesto a analizar: "))

### Obtener los datos del gesto
nombre_tabla = 'raw'

cursor.execute(f"""SELECT fs, onset, abs(CH1) * {reescalado},
                        abs(CH2) * {reescalado}, abs(CH3) * {reescalado}, nombre_gesto
FROM {nombre_tabla}
WHERE gesto_id = ?
AND onset = 1""")
, ( gesto_id ,))

datos = cursor.fetchall ()

nombre_gesto = datos[0][-1] # Obtener el nombre del gesto

fs = datos[0][0] # La frecuencia de muestreo es igual para todos los datos

CH1_values = [dato[2] for dato in datos]
CH2_values = [dato[3] for dato in datos]
CH3_values = [dato[4] for dato in datos]

# Generar el vector de tiempo
N = len(CH1_values)
T = 1.0 / fs # Periodo de muestreo en segundos
time_vector = np.linspace(0.0, N * T, N)

# Eliminar la componente de DC (valor medio)
CH1_values = CH1_values - np.mean(CH1_values)

```

```

CH2_values = CH2_values - np.mean(CH2_values)
CH3_values = CH3_values - np.mean(CH3_values)

# Aplicar el método de Welch para estimar la densidad espectral de potencia
yf_CH1, Pxx_den1 = welch(CH1_values, fs, nperseg=256)
yf_CH2, Pxx_den2 = welch(CH2_values, fs, nperseg=256)
yf_CH3, Pxx_den3 = welch(CH3_values, fs, nperseg=256)

# Graficar los datos
mpl.rc('font', family='Times New Roman')
fig, axs = plt.subplots(3, 1, figsize=(3, 3))

# Límites eje Y
yinf = 0
ysup = 1e-3

# Tamaños de fuente
titulo_size = 13
tick_size = 8
label_size = 9

# Ticks de ejes
#yticks = [0, 0.05, 0.1]
#xticks = [0, 200, 400]

# Gráficos del espectro
plt.subplots_adjust(left=None, bottom=None, right=None, top=None, wspace=None,
                    hspace=None)
axs[0].semilogy(yf_CH1, Pxx_den1, label='CH1')
axs[0].set_ylabel(f'CH1 \n[V^2/Hz]', fontsize = label_size)
axs[0].yaxis.set_major_formatter(mtick.FormatStrFormatter('% .2e'))
axs[0].set_ylim(yinf, ysup)
axs[0].set_title(f'Espectro de Extender \nmuñeca', fontsize = titulo_size)
axs[0].tick_params(labelbottom=False) # Omite los números del eje X

```

```

#axs [0]. set_xticks ( xticks )
#axs [0]. set_yticks ( yticks )
axs [0]. grid ()

axs [1]. semilogy(yf_CH2, Pxx_den2, label='CH2')
axs [1]. set_ylabel (f'CH2\n[V^2/Hz]', fontsize = label_size )
axs [1]. yaxis . set_major_formatter (mtick.FormatStrFormatter(' %.2e'))
axs [1]. set_ylim (yinf , ysup)
axs [1]. tick_params (labelbottom=False) # Omite los números del eje X
#axs [1]. set_xticks ( xticks )
#axs [1]. set_yticks ( yticks )
axs [1]. grid ()

axs [2]. semilogy(yf_CH3, Pxx_den3, label='CH3')
axs [2]. set_ylabel (f'CH3\n[V^2/Hz]', fontsize = label_size )
axs [2]. yaxis . set_major_formatter (mtick.FormatStrFormatter(' %.2e'))
axs [2]. set_ylim (yinf , ysup)
axs [2]. set_xlabel ( 'Frecuencia [Hz]', fontsize = label_size )
#axs [2]. set_xticks ( xticks )
#axs [2]. set_yticks ( yticks )
axs [2]. grid ()
plt . tight_layout ()
fig . subplots_adjust (hspace=0.5, left =0.35)

# Guardar gráfico generado
directorio = 'fig_3ch/welch'
if not os.path . exists ( directorio ):
    os . makedirs( directorio )
nombre_archivo = f"{ gesto_id }_{ nombre_gesto }.png"
ruta_archivo = os . path . join ( directorio , nombre_archivo)
plt . savefig ( ruta_archivo )
print (f"Guardando gráfico en { ruta_archivo }")
#plt . show()

```

```
# Cerrar la conexión a la base de datos
conexion.close ()
```

C.6. Generar la tabla de FFT

, Nombre del archivo: `generar_tabla_fft_gestos.py`

""" FFT de gestos

Script para guardar en una nueva tabla las FFT de los gestos. Incluye funciones para calcular la FFT, RMS, y SNR de los gestos, como también un ejemplo en el que se calcula con todas las entradas de una tabla.

Estructura de la base de datos

```
-----
gesto_id (int)      : Identificador único para cada gesto. Útil para diferenciar
                     distintas instancias del mismo gesto
sesion_id (int)     : ID de la sesión en la que se hizo la captura
nombre_gesto (string) : Nombre del gesto hecho
fecha (string)      : Fecha en la que se hizo la captura, YYYY-MM-DD HH:
                     MM:SS
fs (int)            : Frecuencia de muestreo en Hertz
fc (int)            : Frecuencia de corte del filtro
chX_fft (float)     : FFT de la señal en el canal X
chX_rms_senal (float) : RMS de la señal en el canal X
chX_rms_ruido (float) : RMS del ruido en el canal X
chX_SNR (float)     : SNR del canal X
```

Bastían Rivas

"""

```
import sqlite3
from scipy.fftpack import fft
import os
import matplotlib.pyplot as plt
```

```
# %% Función para calcular RMS
```

```
import numpy as np
```

```

def get_rms(values):
    """
    Calcula el valor RMS de una lista de valores .

    Parameters
    -----
    values ( list or np.array): Lista de valores numéricos.

    Return
    -----
    float : El valor RMS de la lista .
    """
    values = np.array( values )
    rms = np. sqrt ( np.mean(np.square( values )))
    return rms

# %% Función para calcular la SNR
import numpy as np

def get_SNR(signal_rms, noise_rms):
    """
    Calcula la relación señal a ruido (SNR) a partir de los valores RMS de la señal y del
    ruido .

    Parameters
    -----
    signal_rms ( float ): El valor RMS de la señal .
    noise_rms ( float ): El valor RMS del ruido.

    Return
    -----
    float : El valor de la SNR en decibeles (dB).
    """
    snr = 20 * np.log10( signal_rms / noise_rms )

```

```
return snr
```

```
# % %
```

```
def calcular_fft_snr ( gesto_id , ruta_db = 'Datos/ datos_gestos_3ch .db',  
                      tabla_norm = 'norm', fs = 1000, fc = 150,  
                      canales = [1, 2, 3]):
```

```
"""
```

Script para calcular la FFT de hasta tres canales de un gesto específico a partir de su ID, almacenado en una base de datos en SQLite.

Requiere que en la base de datos existan hasta 3 canales de registros del gesto a usar y un registro llamado 'Reposo' correspondiente a la sesión en la que se capturaron los datos

Ojo: retorna el lado derecho de la FFT (aka:mlas frecuencias positivas)

Parameters

- "gesto_id": Número identificador del gesto a normalizar
- "ruta_db": La ruta a la base de datos SQLite que por defecto es 'Datos/ datos_gestos_3ch .db'.
- "tabla_norm": El nombre de la tabla dentro de la base de datos donde se registran los datos previamente filtrados
- "fs": La frecuencia de muestreo, cuyo valor por defecto es 1000 Hz.
- "fc": La frecuencia de corte para el filtrado , cuyo valor por defecto es 150 Hz.
- "canales": Lista de canales a analizar , que por defecto es [1, 2, 3]

Este script realiza las siguientes operaciones:

1. Conectar a la base de datos SQLite especificada .
2. Obtener los datos de la señal para el "gesto_id" proporcionado.
3. Filtrar las señales con la frecuencia y el orden especificado
4. Calcular las envolventes de los 3 canales
5. Normalizar las señales a partir de su contracción voluntaria máxima (CVM)

Return

datos_fft : array_like

Un diccionario con las siguientes entradas :

- gesto_id (int) : Identificador único para cada gesto. Útil para diferenciar distintas instancias del mismo gesto
- sesion_id (int) : ID de la sesión en la que se hizo la captura
- nombre_gesto (string) : Nombre del gesto hecho
- fecha (string) : Fecha en la que se hizo la captura , YYYY-MM-DD HH:MM:SS
- fs (int) : Frecuencia de muestreo en Hertz
- fc (int) : Frecuencia de corte del filtro
- ch1_fft (float) : FFT de la señal en el canal 1
- ch1_rms_senal (float) : RMS de la señal en el canal 1
- ch1_rms_ruido (float) : RMS del ruido en el canal 1
- ch1_SNR (float) : SNR del canal 1
- ch2_fft (float) : FFT de la señal en el canal 2
- ch2_rms_senal (float) : RMS de la señal en el canal 2
- ch2_rms_ruido (float) : RMS del ruido en el canal 2
- ch2_SNR (float) : SNR del canal 2
- ch3_fft (float) : FFT de la señal en el canal 3
- ch3_rms_senal (float) : RMS de la señal en el canal 3
- ch3_rms_ruido (float) : RMS del ruido en el canal 3
- ch3_SNR (float) : SNR del canal 3

''''''

Nuevo: Calcular RMS y SNR de las señales

conexion = sqlite3 .connect(ruta_db)

cursor = conexion.cursor()

```
# Obtener valores de la señal funcional . De acá interesan los valores cuyo onset sea
1, fs, fc, fecha y su nombre
```

```
cursor.execute(f"""
    SELECT sesion_id, fecha, fs, fc, ch1_env_fil , ch2_env_fil , ch3_env_fil ,
        nombre_gesto
    FROM {tabla_norm}
    WHERE gesto_id = ?
    AND onset = 1
""", ( gesto_id , ))
datos = cursor.fetchall ()
```

```
# Formato: [ sesion_id , fs, fc, ch1_env_fil , ch2_env_fil , ch3_env_fil , nombre_gesto]
```

```
# Diccionario para ir registrando los datos que corresponden a cada canal
```

```
canales = [1, 2, 3]
```

```
emg_env_fil = {1: [], 2: [], 3: []}
```

```
for num_canal in canales:
```

```
    # emg_env_fil[i] corresponde al canal i
```

```
    emg_env_fil[num_canal] = [dato[3 + num_canal] for dato in datos]
```

```
# Estos datos son los mismos para los 3 canales a través de todo el tiempo
```

```
sesion_id_gesto = datos[0][0]
```

```
fecha_gesto = datos[0][1]
```

```
fs = datos[0][2]
```

```
fc = datos[0][3]
```

```
nombre_gesto_funcional = datos[0][-1]
```

```
# Obtener los registros de ruido (Reposo) correspondientes a la sesión
```

```
cursor.execute(f"""
    SELECT ch1_env_fil, ch2_env_fil , ch3_env_fil
    FROM {tabla_norm}
    WHERE nombre_gesto LIKE '%Reposo%'
    AND sesion_id = ?
""", ( sesion_id_gesto , ))
```

```

datos = cursor . fetchall ()

ruido_env_fil = {1: [],2: [],3: []}
rms_senal = {1: [],2: [],3: []}
rms_ruido = {1: [],2: [],3: []}
SNR = {1: [],2: [],3: []}
fft_emg = {1: [],2: [],3: []}
fft_emg_magnitude = {1: [],2: [],3: []}

for num_canal in canales :
    # ruido_env_fil [i] corresponde al canal i
    ruido_env_fil [num_canal] = [dato[num_canal - 1] for dato in datos]

    # Calcular RMS de gesto y de ruido
    rms_senal[num_canal] = get_rms( emg_env_fil [num_canal]) # RMS del gesto
    postprocesado
    rms_ruido[num_canal] = get_rms( ruido_env_fil [num_canal]) # RMS del ruido
    postprocesado

    # Obtener SNR con  $SNR = 20 * \log_{10}(RMS\_Señal / RMS\_Ruido)$ 
    SNR[num_canal] = get_SNR(rms_senal[num_canal], rms_ruido[num_canal])

    ### Obtener FFT de la señal
    fft_emg [num_canal] = fft ( emg_env_fil [num_canal])

    # Obtener el lado derecho de la FFT
    N = len(fft_emg [num_canal])
    fft_emg_magnitude[num_canal] = 2.0 / N * np.abs(fft_emg [num_canal][:N//2])

    # Convertir a dB
    fft_emg_magnitude[num_canal] = 20 * np.log10(fft_emg_magnitude[num_canal])

```

```

# Valores a retornar :
resultado_fft = {
    'gesto_id': gesto_id ,
    'sesion_id': sesion_id_gesto ,
    'nombre_gesto': nombre_gesto_funcional ,
    'fecha': fecha_gesto ,
    'fs': fs ,
    'fc': fc ,
    # Canal 1
    'ch1_fft' : fft.emg_magnitude [1], # FFT de la señal en dB
    'ch1_rms_senal' : rms_senal [1],      # RMS de la señal del gesto
    'ch1_rms_ruido' : rms_ruido [1],      # RMS del ruido del canal
    'ch1_SNR' : SNR[1],                  # SNR obtenido a partir del RMS
    # Canal 2
    'ch2_fft' : fft.emg_magnitude [2], # FFT de la señal en dB
    'ch2_rms_senal' : rms_senal [2],      # RMS de la señal del gesto
    'ch2_rms_ruido' : rms_ruido [2],      # RMS del ruido del canal
    'ch2_SNR' : SNR[2],                  # SNR obtenido a partir del RMS
    # Canal 3
    'ch3_fft' : fft.emg_magnitude [3], # FFT de la señal en dB
    'ch3_rms_senal' : rms_senal [3],      # RMS de la señal del gesto
    'ch3_rms_ruido' : rms_ruido [3],      # RMS del ruido del canal
    'ch3_SNR' : SNR[3],                  # SNR obtenido a partir del RMS
}

return resultado_fft

# %%

def registrar_datos_fft ( datos_fft , ruta_db='Datos/ datos_gestos_3ch .db' ,
                        tabla_fft = 'fft' , tabla_norm = 'norm'):
    """
    Registra las FFT calculadas , RMS y SNR en la base de datos .

```

Parameters

- "datos_fft": Datos correspondiente a 1 gesto
- "ruta_db": La ruta a la base de datos SQLite que por defecto es 'Datos/ datos_gestos_3ch .db'.
- "tabla_fft": El nombre de la tabla dentro de la base de datos donde se registran los resultados de la FFT
- "tabla_norm": El nombre de la tabla dentro de la base de datos con los datos normalizados

Estructura de la base de datos

```
gesto_id ( int )      : Identificador único para cada gesto. Útil para
                        diferenciar distintas instancias del mismo gesto
sesion_id ( int )     : ID de la sesión en la que se hizo la captura
nombre_gesto ( string ) : Nombre del gesto hecho
fecha ( string )      : Fecha en la que se hizo la captura ,
                        YYYY-MM-DD HH:MM:SS
fs ( int )            : Frecuencia de muestreo en Hertz
fc ( int )            : Frecuencia de corte del filtro
chX_fft ( float )     : FFT de la señal en el canal X
chX_rms_senal ( float ) : RMS de la señal en el canal X
chX_rms_ruido ( float ) : RMS del ruido en el canal X
chX_SNR (float)       : SNR del canal X
```

```
"""
```

```
conexion = sqlite3 .connect(ruta_db)
```

```
cursor = conexion.cursor()
```

```
# Crear la tabla "fft" y ejecutar la consulta para crearla
```

```
cursor .execute(f"""
```

```
CREATE TABLE IF NOT EXISTS {tabla_fft} (
```

```
    id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```

        gesto_id INTEGER,
        sesion_id INTEGER,
        nombre_gesto TEXT,
        fecha TEXT,
        fs INTEGER,
        fc INTEGER,
        ch1_fft REAL,
        ch1_rms_ruido REAL,
        ch1_rms_senal REAL,
        ch1_SNR REAL,
        ch2_fft REAL,
        ch2_rms_ruido REAL,
        ch2_rms_senal REAL,
        ch2_SNR REAL,
        ch3_fft REAL,
        ch3_rms_ruido REAL,
        ch3_rms_senal REAL,
        ch3_SNR REAL

    );
    """

```

```

# Anotar todos los registros correspondientes al gesto especificado
# Los 3 canales tienen el mismo largo así que se puede usar cualquiera
n_registros = len( datos_fft [ ' ch1_fft ' ])

```

```

for i in range( n_registros ):
    insertar_query = f"""
    INSERT INTO {tabla_fft}( gesto_id , sesion_id , nombre_gesto, fecha, fs, fc,
                                ch1_fft , ch1_rms_ruido, ch1_rms_senal, ch1_SNR,
                                ch2_fft , ch2_rms_ruido, ch2_rms_senal, ch2_SNR,
                                ch3_fft , ch3_rms_ruido, ch3_rms_senal, ch3_SNR)

```

```

VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
""""

valores = (
    datos_fft [ ' gesto_id ' ],
    datos_fft [ ' sesion_id ' ],
    datos_fft [ ' nombre_gesto ' ],
    datos_fft [ ' fecha ' ],
    datos_fft [ ' fs ' ],
    datos_fft [ ' fc ' ],
    datos_fft [ ' ch1_fft ' ][ i ],
    datos_fft [ ' ch1_rms_ruido ' ],
    datos_fft [ ' ch1_rms_senal ' ],
    datos_fft [ ' ch1_SNR ' ],
    datos_fft [ ' ch2_fft ' ][ i ],
    datos_fft [ ' ch2_rms_ruido ' ],
    datos_fft [ ' ch2_rms_senal ' ],
    datos_fft [ ' ch2_SNR ' ],
    datos_fft [ ' ch3_fft ' ][ i ],
    datos_fft [ ' ch3_rms_ruido ' ],
    datos_fft [ ' ch3_rms_senal ' ],
    datos_fft [ ' ch3_SNR ' ],

)

cursor.execute( insertar_query , valores )

conexion.commit()
conexion.close ()

print (f"Registrado '{ datos_fft [ ' nombre_gesto ' ] }' con ID
      { datos_fft [ ' gesto_id ' ] } en la tabla '{ tabla_norm }'." )

# %%

```

```

if __name__ == '__main__':
    '''
    Generar una tabla con la FFT de TODOS los datos normalizados registrados en
    tabla_norm.
    Se appoya en las funciones de registrar_datos_fft y calcular_fft_snr .
    También sirve como ejemplo de uso para las funciones anteriormente
    mencionadas
    '''

    ruta_db = '3 ch_gestos_testing .db'
    tabla_norm = 'norm'
    tabla_fft = 'fft'

    # Conectarse a la base de datos
    conexion = sqlite3 .connect(ruta_db)
    cursor = conexion.cursor()

    # Mostrar la ubicación de la base de datos en consola
    print(f"Usando base de datos en: {os.path.abspath(ruta_db)}")
    print(f"Los gestos normalizados se leerán en la tabla '{tabla_norm}' y las
          FFT se guardarán en '{tabla_fft}'")

    conexion = sqlite3 .connect(ruta_db)
    cursor = conexion.cursor()

    # Inicio de interacción con usuario
    # Mostrar los gestos almacenados en la tabla de los normalizados
    cursor.execute(f"""
        SELECT gesto_id, MIN(fecha), nombre_gesto, sesion_id
        FROM {tabla_norm}
        GROUP BY gesto_id
    """)
    datos = cursor . fetchall ()

```

```

    print (f"----- Gestos registrados ----
ID \tFecha          \tSesión\tNombre del gesto
-----\t-----\t-----\t-----")

    gestos_a_procesar = []
    for fila in datos:
        gesto_id, fecha, gesto, sesion_id = fila
        print (f"{ gesto_id }\t{ fecha }\t{ sesion_id }\t{ gesto }")
        gestos_a_procesar.append(gesto_id)

    n_gestos = len( gestos_a_procesar )
    rpta = []

    # Confirmación porque puede tomar un rato
    while rpta not in ["y", "n"]:
        rpta = input(f"¿ Calcular la FFT de los { n_gestos } gestos?
[Y/n]: ").lower()

    # Empezar con procesamiento si se recibió una Y
    if rpta == "n":
        print ("Cancelando ... ")
        quit ()
    else:
        for gesto in gestos_a_procesar:
            # Obtener FFT de todos los gestos
            datos_fft = calcular_fft_snr (gesto, ruta_db, tabla_norm)
            registrar_datos_fft ( datos_fft , ruta_db , ' fft ')

    print (f"Finalizado . { n_gestos } gestos registrados .")

    conexion.close ()

```

C.7. Calcular la desviación estándar

Nombre del archivo: `calc_stddev_snr.py`

```
''' Calcular desviación estándar
```

Este script auxiliar está hecho para compensar por la falta de una función que permita calcular desviación estándar usando una query en SQLite.

Bastián Rivas

```
'''
```

```
import sqlite3
```

```
import pandas as pd
```

```
import numpy as np
```

```
# Conectar a la base de datos
```

```
conexion = sqlite3.connect('Datos/ datos_gestos_3ch.db')
```

```
# Definir una consulta para obtener los datos necesarios
```

```
query = '''
```

```
SELECT nombre_gesto, sesion_id, ch1_SNR, ch2_SNR, ch3_SNR
```

```
FROM fft
```

```
WHERE sesion_id IN (1, 2, 3, 4, 5, 6, 7);
```

```
'''
```

```
# Ejecutar la consulta y cargar los datos en un DataFrame de Pandas
```

```
df = pd.read_sql_query(query, conexion)
```

```
# Agrupar los datos según el nombre de cada gesto
```

```
agrupados_por_gesto = df.groupby('nombre_gesto')
```

```
# Calcular la desviación estándar para cada gesto y cada canal de SNR
```

```
resultados = agrupados_por_gesto.agg({
```

```
    'ch1_SNR': lambda x: round(np.std(x), 2),
```

```
    'ch2_SNR': lambda x: round(np.std(x), 2),
```

```
'ch3_SNR': lambda x: round(np.std(x), 2),
}).reset_index ()

# Cerrar la conexión a la base de datos
conexion.close ()

# Mostrar los resultados
print ( resultados )
```