ULPGC
Universidad de
Las Palmas de
Gran Canaria

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

DESARROLLO DE UNA PLATAFORMA DE BAJO COSTE PARA EL RECONOCIMIENTO DE GESTOS BASADO EN ELECTROMIOGRAFÍA

Titulación: Grado en Ingeniería en Tecnologías de la Telecomunicación

Mención: Sistemas Electrónicos

Autor/a: Laura Castro Vega

Tutor/a: D. Valentín De Armas Sosa

D. Félix B. Tobajas Guerrero

Fecha: Julio 2022

Agradecimientos

En primer lugar, quiero mencionar a mis padres y a toda mi familia, por su paciencia y apoyo incondicional durante estos años difíciles. Por brindarme la posibilidad de formarme con una educación de calidad y guiarme en el camino para labrar un buen fututo personal y profesional.

También, a los compañeros y amigos con los que he tenido el placer de coincidir en esta etapa. Desde aquellos con los que empecé hasta los que me acogieron y sustentaron en las etapas de cambio. Por compartir experiencias y alegrarse de mis logros como si fuesen los suyos propios.

Por último, me gustaría agradecer a mis tutores Félix B. Tobajas Guerrero y Valentín de Armas Sosa la oportunidad de realizar este Trabajo Fin de Grado junto a ellos, por su total implicación y por su apoyo, esfuerzo y contribución durante la realización del proyecto.

Resumen

Los sistemas inalámbricos han ido integrándose en los últimos años en las soluciones IoT (*Internet of Things*) con el doble reto de reducir el coste y el consumo de los dispositivos. Los avances tecnológicos en la fabricación de dispositivos electrónicos inteligentes de altas prestaciones han permitido contribuir a la mejora de la calidad de vida de los usuarios, a pesar de implicar estos grandes retos. Por ello, con el desarrollo de este Trabajo Fin de Grado se consigue un dispositivo de bajo coste, portátil y personalizable que puede utilizarse en campos IoT. Para ello, se plantea el desarrollo de una plataforma *hardware/software* de bajo coste que se utiliza para el reconocimiento basado en electromiografía de gestos de la mano. Se utiliza el dispositivo *Myo Armband* para la detección del movimiento de la mano, cuya información se envía de forma inalámbrica mediante la tecnología BLE (*Bluetooth Low Energy*) a un dispositivo de IoT basado en MCU (*MicroController Unit*). Se consigue un dispositivo fácilmente adaptable al usuario al integrar todo el proceso de entrenamiento y clasificación SVM (*Support Vector Machine*) en la plataforma desarrollada.

En primer lugar, se lleva a cabo una etapa de estudio donde se ahonda en la tecnología de comunicación inalámbrica BLE, conceptos básicos de *Machine Learning*, métodos de clasificación y el algoritmo SVM, al ser aspectos relevantes para el desarrollo del TFG. Posteriormente, se efectúa un estudio de los parámetros asociados a las señales EMG con el fin de conseguir una clasificación óptima de los datos obtenidos con el dispositivo *Myo*. Finalmente, se desarrolla la plataforma objetivo, incorporando de manera progresiva las distintas funcionalidades que dispone la plataforma *hardware/software* final, detallando las librerías y los componentes *hardware* empleados y analizando las pruebas llevadas a cabo para la validación de su funcionamiento.

Abstract

In recent years, Wireless systems have been integrated into Iot (Internet of Things) solutions with the dual challenge of reducing the cost and power of devices. Technological advances in the development of high performance smart electronic devices have contributed to the improvement of the quality of life of its users, despite involving the mentioned challenges. Therefore, the main goal of this Final Degree Project is to develop a low-cost, portable, and customizable device that can be used in IoT applications. For this purpose, the proposal is the development of a low-cost hardware/software platform used for electromyography-based recognition of hand gestures. The Myo Armband device is used for hand movement detection, whose information is sent wirelessly via BLE (Bluetooth Low Energy) technology to an IoT based on MCU (MicroController Unit) device. A device easily adaptable to the user is achieved by integrating the entire SVM (Support Vector Machine) training and classification process in the developed platform.

First, a study of relevant aspects for the Final Degree Project is presented analyzing the BLE wireless communication technology, basic concepts of Machine Learning, classification methods and the SVM algorithm. Subsequently, a study of the parameters associated with EMG signals is explained to achieve an optimal classification of the data obtained with the Myo device. Finally, the target platform is developed, with the progressive incorporation of the different functionalities that the final hardware/software platform will have, detailing the libraries and hardware components used, and analyzing the test carried out for the validation of its operation.

Contenido

Índice de tablas

Índice de figuras

Índice de acrónimos

ADC *Analog-to-Digital Converter*

AI *Artificial Intelligence*

ASCII *American Standard Code for Information Interchange*

ATT *Attribute Protocol*

BLE *Bluetooth Low Energy*

BBR *Bluetooth Basic Rate*

BEDR *Bluetooth Enhanced Data Rate*

BR/EDR *Basic Rate/Enhanced Data Rate*

CMOS *Complementary Metal Oxide Semiconductor*

COITT *Colegio Oficial de Ingenieros Técnicos de Telecomunicación*

CRC *Cyclic Redundancy Check*

DRAM *Dynamic Random Access Memory*

EEPROM *Electrically Erasable Programmable Read-Only Memory*

EITE *Escuela de Ingeniería de Telecomunicación y Electrónica*

EMG *Electromiografía*

FFT *Fast Fourier Transform*

FRAM *Ferroelectric Random Access Memory*

GAP *Generic Access Profile*

GATT *Generic Attribute*

GPS *Global Positioning System*

HCI *Host Controller Interface*

HMI *Human-Machine Interface*

I2C *Inter-Integrated Circuit*

IDE *Integrated Development Environment*

IMU *Inertial Measurement Unit*

IoT *Internet of Things*

ISM *Industrial, Scientific, and Medical*

IUMA *Instituto Universitario de Microelectrónica Aplicada*

L2CAP *Logical Link Control and Adaptation Protocol*

LCD *Liquid Crystal Display*

LED *Light Emitting Diode*

LIBSVM *Library for Support Vector Machines*

LL *Link Layer*

MCU *MicroController Unit*

MIC *Message Integrity Check*

MUAP *Motor Unit Action Potencial*

MUAPT *Motor Unit Action Potencial Train*

NFC *Near Field Communication*

OLED *Organic Light-Emitting Diode*

OSI *Open System Interconnection*

PDU *Protocol Data Unit*

PHY *Physical Layer*

PWM *Pulse Width Modulation*

RAM *Random Access Memory*

RBF *Radial Basic Function*

RMS *Root Medium Square*

SDIO *Security Digital Input Output*

SIG *Special Interest Group*

SMP *Single Management Protocol*

SRAM *Static Random Access Memory*

SSH *Secure Shell*

SVC *Support Vector Classification*

SVM *Support Vector Machines*

SVR *Support Vector Regression*

TCP *Transmission Control Protocol*

Telnet *Telecommunication Network*

UART *Universal Asynchronous Receiver Transmitter*

ULPGC *Universidad de Las Palmas de Gran Canaria*

USB *Universal Serial Bus*

UUID *Universally Unique Identifier*

Wi-Fi *Wireless Fidelity*

CAPÍTULO 1 Introducción

En este capítulo se presenta el planteamiento del problema a abordar en el presente Trabajo Fin de Grado, así como los objetivos que se pretenden alcanzar durante su desarrollo. Por otra parte, para establecer un marco teórico en el que enmarcar la motivación del presente TFG y el uso del dispositivo *Myo Armband,* se presentan algunos de los campos de aplicación más importantes y los últimos avances tecnológicos en aspectos relacionados con el mismo. Además, se incluye el peticionario y se desglosa la estructura del documento con el objetivo de tener una visión general del desarrollo del trabajo realizado.

1.1 Antecedentes

Desde que entre los años 2008 y 2009 nace el término *Internet of Things* (IoT) [1], es más que evidente que esta arquitectura emergente ha introducido una gran cantidad de nuevas oportunidades en el acceso de datos a servicios específicos en la educación, en seguridad, transporte o asistencia sanitaria, entre otros muchos campos. Aprovechando estos avances es posible conseguir importantes mejoras en la calidad de vida de las personas, pudiendo destacar aquellas con necesidades especiales, como personas mayores, enfermas o con alguna discapacidad. Poder detectar estados a través de sensores y realizar una correcta y detallada obtención y procesamiento de datos permite dar respuesta a los cambios que se producen en el mundo real. Es, por todo esto, que puede decirse que uno de los campos en los que IoT ha adquirido una gran presencia es en el ámbito de la accesibilidad. Además, los avances tecnológicos en la fabricación de dispositivos electrónicos inteligentes de altas prestaciones, cada vez más disponibles a menor coste, permiten contribuir también a la mejora de la calidad de vida de los usuarios en múltiples ámbitos de aplicación.

Por otra parte, otro de los campos destacables de IoT se encuentra en las tecnologías inalámbricas y las interfaces que permiten que los dispositivos intercambien datos a través de una red sin necesidad de una infraestructura cableada o de una serie de periféricos. Por todo esto, los beneficios que subyacen en Internet de las Cosas son muy diversos y amplios, pero hacerlos realidad implica importantes retos. Aspectos como el consumo, coste o autonomía de las baterías de los dispositivos son elementos importantes a tener en cuenta para la implementación de cualquier proyecto de IoT. Todas estas aplicaciones necesitan maximizar la autonomía de las baterías con el fin de ahorrar costes, mejorar la experiencia de usuario y cumplir con los requisitos normativos.

Con la propuesta que se plantea para este Trabajo Fin de Grado se pretende abordar los problemas o dificultades encontrados y los antecedentes planteados. Se plantea el desarrollo de una plataforma de bajo coste para el reconocimiento de gestos mediante electromiografía. Para ello se ha optado por realizar un estudio de la detección de gestos de la mano mediante el dispositivo *Myo Armband* [2][3] que permite su uso como interfaz en aplicaciones de IoT. Este dispositivo fue desarrollado y lanzado en 2015 por la empresa *Thalmic Labs*, refundada como *North*. La estructura física del brazalete a utilizar se muestra en la Figura 1. Una de las ventajas que se conseguirán es la escalabilidad, ya que podría reutilizarse con diferentes elementos que también sean capaces de capturar la información, adaptándola de forma adecuada en el dispositivo. Además, se pretende implementar una solución de bajo consumo, por lo que BLE (*Bluetooth Low Energy*) será la tecnología escogida para la comunicación inalámbrica entre dispositivos [4].

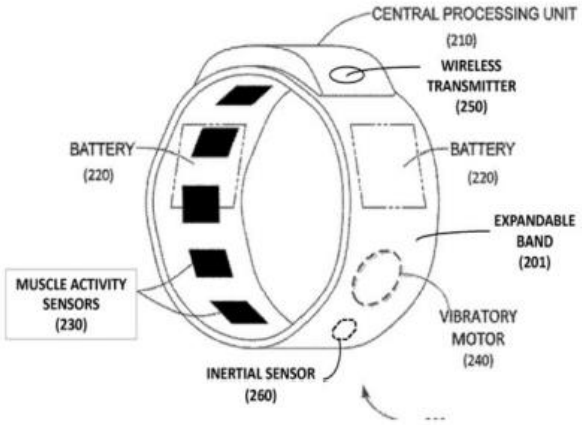

Figura 1. Estructura física del brazalete Myo Armband [2]

Para la clasificación de los gestos de la mano se optará por el método de clasificación SVM (*Support Vector Machine*), ya que presenta un conjunto de algoritmos de aprendizaje supervisado. Está basado en una primera fase de entrenamiento donde, a partir de un conjunto de muestras, se pueden etiquetar distintas clases, y una segunda fase en la que se construye un modelo capaz de predecir la clase de una nueva muestra. Para el desarrollo del proceso de clasificación SVM se empleará como referencia el *software* LIBSMV, una librería que implementa diferentes formulaciones de SVM para la clasificación, regresión y estimación de distribución. Por tanto, el propósito final consistirá en integrar en un único dispositivo de bajo coste basado en MCU (*MicroController Unit*) para aplicaciones de IoT todas las fases de SVM, sin necesidad de recurrir a un ordenador. Un esquema general del planteamiento realizado se muestra en la Figura 2. El brazalete *Myo* será el encargado de capturar los datos relativos a la posición de la mano al realizar algún gesto, que serán transferidos de forma inalámbrica a un dispositivo de

IoT basado en MCU, encargado de realizar el procesamiento, entrenamiento y clasificación correspondiente para finalmente mostrar, por ejemplo, a través de una pantalla LCD (*Liquid Crystal Display*), el gesto identificado.

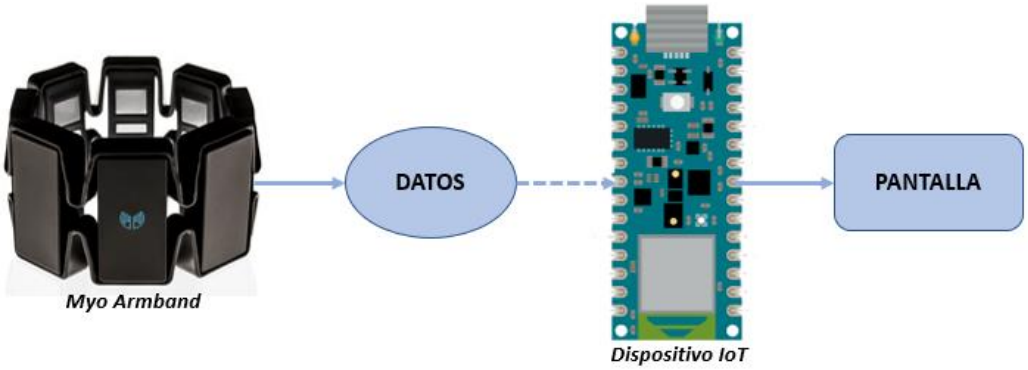

Figura 2. Planteamiento general de la plataforma

Para la comunicación con el dispositivo *Myo Armband* se tomará como referencia inicial el Trabajo Fin de Máster titulado "Plataforma para la interpretación del alfabeto dactilológico de Lengua de Signos Española basada en el dispositivo *Myo Armband*", realizado por D. Carlos Santiago Viera Betancor en el Instituto Universitario de Microelectrónica Aplicada (IUMA) de la ULPGC.

1.2 Campos de aplicación del dispositivo *Myo Armband* y últimos avances tecnológicos

La creciente integración entre la tecnología y el ser humano es cada vez más notable y, por este motivo, en los últimos años ha aumentado significativamente el número de estudios sobre equipos electrónicos flexibles, plegables y portátiles. Por ello, multitud de investigadores y empresas buscan desarrollar dispositivos con altas prestaciones, más sofisticados y complejos, pero sencillos de utilizar para los usuarios. Por todo eso, resulta interesante destacar el uso del dispositivo *Myo Armband*. Aplicado al campo protésico, este dispositivo portátil permite solucionar numerosos problemas existentes con prótesis ya existentes, presentando una plataforma electrónica completa que detecta en tiempo real las principales señales relacionadas con la actividad del antebrazo y siendo capaz de transmitirlas a otros dispositivos conectados [5]. Actualmente, es posible encontrar prótesis de bajo coste, livianas y compactas que permiten un número limitado de movimientos, y prótesis de alta gama, más complejas, caracterizadas por su destreza, voluminosas y de mayor coste. El brazalete *Myo* plantea una solución óptima y supone un compromiso entre ambas opciones al considerarse una propuesta de bajo coste con la que se consigue un sistema confiable y fácil de usar.

El uso de este dispositivo, así como el estudio y análisis de las señales EMG está a la orden del día. Existen numerosos estudios de investigación sobre posibles aplicaciones avanzadas del brazalete *Myo* que demuestran la eficacia y el potencial del dispositivo. Uno de los estudios se centró en la implementación de una plataforma llamada *Myo HMI (Human-Machine Interface)* centrada en el desarrollo de una interfaz gráfica que brindase a los usuarios la posibilidad de visualizar en tiempo real en un PC las señales mioeléctricas proporcionadas por el brazalete según los movimientos del brazo [6]. Además, se han desarrollado también prótesis de mano impresa en 3D y sistemas de realidad virtual, con el objetivo de mostrar una mano humana virtual en la interfaz y verificar la funcionalidad del *software Myo HMI*. Otros estudios han permitido obtener datos muy claros y relevantes de los músculos del brazo de los usuarios que han sido utilizados para controlar brazos robóticos [7]. El objetivo de los investigadores era utilizar estos robots para llegar a entornos bioinfectados o radioactivos sin necesidad de presencia humana, así como de ayudar a personas mayores o discapacitadas en tareas de movimiento. De esta manera, se ha determinado que el dispositivo *Myo* permite obtener una respuesta de control muy rápida con una tasa media de clasificación correcta de los movimientos.

Sin embargo, el dispositivo *Myo Armband* no limita su uso en el campo protésico, pudiendo utilizarse en numerosas aplicaciones gracias al control de los movimientos naturales de la mano. Un campo de aplicación bastante extendido del brazalete es el de los juegos y el entretenimiento. Puede emplearse para navegar entre pestañas o ventanas de un ordenador, controlar el paso de diapositivas en una presentación, manejar sistemas de sonido o iluminación, etc. También, puede utilizarse para controlar drones o realizar movimientos con robots. Aparte de estos campos, también encuentra aplicación en el ámbito sanitario. Existen empresas que ya han conseguido integrar el brazalete *Myo*, junto con varias cámaras y un *software* de reconocimiento de voz para desarrollar una tecnología dirigida a los cirujanos que permite que estos profesionales puedan controlar en tiempo real varios instrumentos médicos durante las cirugías [8]. Además, ofrece un enorme potencial para el control de los movimientos de una prótesis que reemplaza una extremidad amputada, permitiendo realizar movimientos precisos y correctos.

En definitiva, el brazalete *Myo Armband* presenta muchas ventajas en comparación con otros dispositivos o sensores utilizados para adquirir la actividad de los músculos al integrar tanto los electrodos EMG como la IMU para la detección de la posición de la mano en el espacio tridimensional. Además, supone una solución fácil para el usuario, ya que su colocación es

sencilla y requiere de una instalación de sensores EMG en la piel. Esto lo hace un dispositivo compacto que permite la correcta detección de las señales de forma inalámbrica, utilizando la tecnología BLE. Por todo esto, este brazalete se ajusta a las necesidades requeridas para el desarrollo del presente TFG y ha sido elegido como dispositivo encargado de capturar los datos inicialmente.

1.3 Objetivos

El objetivo principal de este Trabajo Fin de Grado consiste en el desarrollo de una plataforma *hardware/software* de bajo coste que se utilizará para el reconocimiento de gestos de la mano basado en electromiografía. Para la detección del movimiento de la mano se utilizará el dispositivo *Myo Armband*, que cuenta con ocho sensores electromiográficos (EMG), un acelerómetro, un giroscopio y un magnetómetro. La transmisión de la información asociada a los gestos de la mano se realizará de forma inalámbrica, mediante conexión BLE.

Asimismo, a diferencia de otros proyectos, se pretende conseguir integrar todo el proceso de clasificación SVM en un único dispositivo de IoT basado en MCU (*MicroController Unit*), en concreto el modelo *Arduino Nano 33 BLE Sense*. De esta manera, se pretende conseguir reducir costes, al no necesitar de un PC para el entrenamiento, y aportar muchas ventajas asociadas a la obtención de un dispositivo totalmente personalizable y portátil. El diagrama que representa la idea plantada para el desarrollo de este Trabajo Fin de Grado se muestra en la Figura 3.

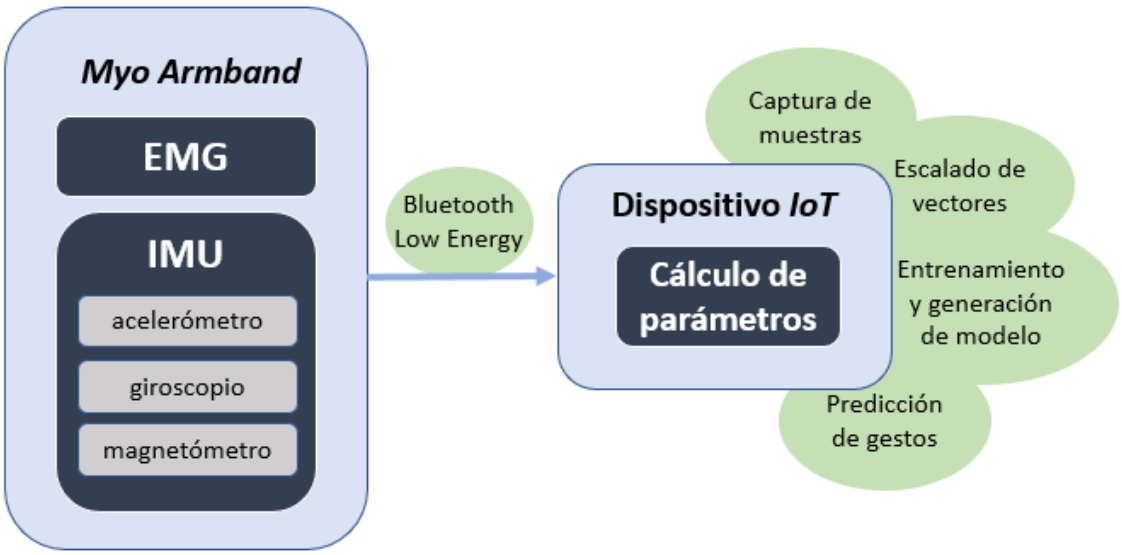

Figura 3. Diagrama de la plataforma objetivo planteada

Este objetivo general se desglosa en los siguientes objetivos específicos:

Objetivo 1 (O1). Detección de símbolos o gestos mediante un entrenamiento con los datos obtenidos a través del dispositivo *Myo Armband*. Mediante la conexión BLE con el dispositivo *Arduino Nano 33 BLE Sense*, y extraer la información de los sensores para su posterior procesamiento.

Objetivo 2 (O2). Clasificación de los símbolos detectados basada en SVM mediante el desarrollo del código necesario para que el dispositivo *Arduino* procese los datos del brazalete.

Objetivo 3 (O3). Integración de todo el proceso en un único dispositivo de IoT personalizable, y representación de los gestos detectados.

1.4 Peticionario

Actúa como peticionario del presente Trabajo Fin de Grado (TFG) la Escuela de Ingeniería de Telecomunicación y Electrónica (EITE) de la Universidad de las Palmas de Gran Canaria (ULPGC) como requisito indispensable para la obtención de título de Graduada en Ingeniería en Tecnologías de la Telecomunicación, tras haber superado con éxito las asignaturas especificadas en el Plan de Estudios.

1.5 Estructura del documento

Este documento se encuentra dividido en tres partes diferenciadas: Memoria, Pliego de Condiciones y Presupuesto. A su vez, la Memoria se ha estructurado en siete capítulos, además de incluir las referencias bibliográficas empleadas. A continuación, se presenta una breve descripción de los principales aspectos que tratan los diferentes capítulos.

Capítulo 1. Introducción. En este capítulo se plantea el problema que ha motivado la realización de este Trabajo Fin de Grado y se exponen las soluciones que se van a desarrollar, detallando los objetivos propuestos. Además, se plantean aspectos relevantes sobre las aplicaciones del dispositivo inalámbrico *Myo Armband,* con el fin de comprender la motivación de su uso para el desarrollo de la plataforma. Por último, se incluye el peticionario y se plantea la estructura de la Memoria para ofrecer una visión global del trabajo realizado.

Capítulo 2. Protocolo BLE. Se presenta una introducción a la tecnología de comunicación inalámbrica *Bluetooth Low Energy* (BLE). Se definen los principales aspectos de su arquitectura,

topología de red, paquetes de datos, servicios y características, para disponer de una base de conocimiento de la tecnología de comunicación empleada en la plataforma.

Capítulo 3. Proceso de clasificación. En este capítulo se plantean los conceptos básicos de *Machine Learning* y se tratan diferentes métodos de clasificación. Se aborda en detalle el algoritmo SVM, al ser el método de clasificación empleado en el presente TFG para la identificación de los gestos de la mano. Por último, se introduce el paquete *software* LIBSVM, que sustenta las bases del clasificador integrado en la plataforma.

Capítulo 4. Parámetros y señales electromiográficas. Se realiza una introducción teórica sobre las señales electromiográficas, al considerarse una parte importante en la obtención de los datos con el dispositivo *Myo*. Se detalla el estudio de los diferentes parámetros asociados a estas señales EMG para la búsqueda de una clasificación óptima de los datos, además de presentarse las pruebas realizadas en base a estos parámetros.

Capítulo 5. Desarrollo de la plataforma *hardware/software* inicial. En este capítulo se presentan los desarrollos iniciales, con el objetivo de reflejar el procedimiento llevado a cabo hasta la obtención del resultado final, incorporando progresivamente funcionalidad a la plataforma. Se detalla la librería empleada para poder establecer la comunicación BLE con el dispositivo *Myo*, así como los componentes *hardware* que se emplean. Finalmente se describe el *firmware* desarrollado y se analizan las pruebas llevadas a cabo para validar su funcionamiento.

Capítulo 6. Desarrollo de la plataforma *hardware/software* final. En este capítulo se presentan los pasos seguidos para lograr la integración de la funcionalidad completa en la plataforma final. Se plantean los cambios de *hardware* realizados, con su correspondiente justificación, además de analizar la adaptación del código LIBSVM a la nueva implementación en el dispositivo basado en MCU. También, se introducen las nuevas funcionalidades de la plataforma, y se analizan las pruebas llevadas a cabo para validar su funcionamiento.

Capítulo 7. Conclusiones. Se plantean las conclusiones obtenidas tras haber cumplimentado las tareas y objetivos propuestos para el presente Trabajo Fin de Grado.

Finalmente, se incluye la bibliografía empleada para la realización del TFG, agrupando las referencias y documentos que han sido consultados durante el desarrollo del trabajo. Posteriormente, siguiendo la estructura del documento, se encuentra el Pliego de Condiciones y el Presupuesto.

Pliego de Condiciones. Se exponen las condiciones *hardware* y *software* con las que se ha desarrollado el TFG.

Prepuesto. Se incluyen los gastos generados en la realización del presente TFG.

CAPÍTULO 2 Protocolo BLE

En el presente TFG se pretende proporcionar una solución de bajo consumo, por lo que para el desarrollo de la plataforma *hardware/*software que se plantea se utiliza la tecnología *Bluetooth Low Energy* para la comunicación inalámbrica entre el dispositivo *Myo Armband* y la plataforma de IoT basada en MCU. En este capítulo se presentan las características principales de este protocolo de comunicación y se realiza una descripción de la tecnología para conocer sus prestaciones y comprender su funcionamiento.

2.1 ¿Qué es *Bluetooth Low Energy*?

La tecnología *Bluetooth* surge a principios de los años 2000 con el objetivo de permitir el intercambio de datos de forma inalámbrica entre dos dispositivos sin necesidad de ningún otro equipo de red intermedio. La primera versión de la tecnología *Bluetooth* se conoce como *Bluetooth Basic Rate* (BBR) y ofrecía una velocidad de datos bruta en la capa física de 1 Mbps. Posteriormente se definió una versión más rápida conocida como *Bluetooth Enhanced Data Rate* (BEDR) que aumentaba la velocidad a 2 Mbps [9]. Aparece, posteriormente, como alternativa a BEDR, *Bluethooth Low Energy* (BLE), presentando unas capacidades y cualidades que la hacen perfecta para una nueva generación de productos, cumpliendo con exigentes requisitos técnicos y funcionales.

Bluethooth Low Energy es una tecnología inalámbrica de baja potencia utilizada para la conexión de dispositivos. BLE está orientada a aplicaciones que requieren consumir menos energía y necesitan operar con baterías durante periodos de tiempo prolongados. Fue desarrollada originalmente por Nokia bajo el nombre de *Wibree,* pero luego pasó a ser adoptada en el año 2010 por *Bluetooth Special Interest Group* (SIG) como parte de la especificación *Bluetooth* 4.0. Sin embargo, no se considera una actualización del *Bluetooth* original, sino una nueva tecnología que utiliza la marca *Bluetooth* y que se enfoca más en Internet de las Cosas o aplicaciones en las que pequeñas cantidades de datos se transfieren a velocidades más bajas. Su principal objetivo es diseñar un estándar de radio con el menor consumo de energía posible, especialmente optimizado para tener un bajo coste, con bajo ancho de banda, reducida potencia y complejidad [10]. BLE opera en la banda de frecuencia ISM (*Industrial, Scientific and Medical*) de 2,4 GHz, que es el mismo espectro que utiliza el *Bluetooth* clásico y Wi-Fi (*Wireless Fidelity*), entre otras tecnologías. Las bandas de radio ISM están reservadas internacionalmente para el uso no comercial de radiofrecuencia electromagnética en áreas industrial, científica y médica.

La versión de especificaciones 4.0 de *Bluetooth* define dos tecnologías inalámbricas: *Bluetooth* clásico (BR/EDR, *Basic Rate/Enhanced Data Rate*) y BLE. Ambos estándares de comunicación no son directamente compatibles entre sí, de forma que cualquier dispositivo con una versión de *Bluetooth* anterior a la 4.0 no podrá comunicarse directamente de ninguna manera con un dispositivo BLE. A su vez, existen dos tipos de dispositivos que pueden ser utilizados con estas configuraciones [11]. En primer lugar, dispositivos modo doble, que implementan ambas configuraciones BLE y BR/EDR y pueden comunicarse con cualquier dispositivo *Bluetooth*, y en segundo lugar, dispositivos en modo simple, que utilizan el estándar BLE y pueden comunicarse tanto con otros dispositivos modo simple o con dispositivos modo doble, pero no con dispositivos que soporten únicamente BR/EDR.

BLE hace uso de dos tipos de dispositivos, uno central y un periférico. El dispositivo central es aquel que presenta mejores prestaciones en cuanto a potencia, memoria o capacidad de batería. Sin embargo, los periféricos cuentan con recursos mucho más limitados, especialmente cuando se trata de la batería. Esto significa que BLE es una tecnología asimétrica, donde la gran parte del trabajo pesado del procesamiento lo sustenta el dispositivo central. Esto permite que el periférico pueda permanecer inactivo durante periodos de tiempo más largos, consumiendo menos potencia. De esta forma, los dispositivos están activos únicamente cuando se requiere la transmisión de datos. Por tanto, la forma en la que BLE logra su consumo de energía optimizado se basa en el apagado de las radios de los dispositivos periféricos, tanto como sea posible, y en el envío de pequeñas cantidades de datos a velocidades de transferencia pequeñas. No está pensado para mantener conexiones entre dispositivos durante un largo periodo de tiempo transmitiendo grandes cantidades de datos a alta velocidad. Por eso, aplicaciones como la transmisión de vídeo en vivo o de audio de alta calidad, así como la transferencia de grandes cantidades de datos, no son realmente adecuadas para BLE. En cambio, sí que resulta óptimo para la transferencia de datos desde dispositivos pequeños como sensores a teléfonos inteligentes u otros dispositivos para el procesamiento de la información.

El rango de alcance posible entre dos dispositivos BLE se encuentra dentro de los 50 metros. Sin embargo, si existen obstáculos o paredes entre los dispositivos el rango puede disminuir considerablemente. A pesar de esto, existen tres importantes factores que pueden limitar el alcance entre dispositivos BLE: el entorno, el diseño de la antena del dispositivo, y la orientación entre los terminales que se comunican.

En cuanto a la velocidad de BLE, existen dos conceptos importantes a tener en cuenta. Uno es la tasa de datos sin procesar y el otro es la tasa de datos de la aplicación. La primera es

la tasa a la que la radio transmite los datos por aire y varía en función de la versión de *Bluetooth* que se esté utilizando. Por ejemplo, si la versión utilizada es la 4.2, la velocidad de datos sin procesar se establece en 1 Mbps. Sin embargo, si se emplea la versión 5.0 esta tasa puede variar, existiendo diferentes opciones. Así, puede usar 1 Mbps, que es la velocidad de datos de baja energía original, puede aumentar hasta 2 Mbps cuando se utiliza la función de mayor velocidad de *Bluetooth* 5.0, o reducirse a 500 Kbps, o incluso 125 Kbps, cuando se utiliza la función de largo alcance. Por otra parte, en términos de tasa de datos de aplicación, esta se ve reducida debido a la sobrecarga de los paquetes y a la existencia de determinadas brechas entre la transmisión de algunos de los paquetes. Utilizando una tasa de datos brutos de 2 Mbps, se puede conseguir hasta 1.4 Mbps de tasa de datos de aplicación.

Las aplicaciones más comunes de BLE están relacionadas con tareas de automatización del hogar (cerraduras inteligentes, sistemas de iluminación…), dispositivos portátiles, tecnologías de ubicación de interiores donde GPS puede no ser factible, dispositivos médicos o dispositivos de salud personal. Por tanto, BLE se presenta como una opción muy factible, especialmente cuando se trata de domótica y aplicaciones industriales. Por tanto, el uso de BLE supone una serie de beneficios. Por ejemplo, su bajo consumo de energía, su coste gratuito para acceder a las especificaciones oficiales de la tecnología, los costes de los chips o módulos BLE, o su presencia en la mayoría de los teléfonos móviles inteligentes del mercado, lo que supone una gran ventaja sobre otras tecnologías.

2.2 Arquitectura BLE

BLE está formado por diferentes capas que interactúan entre sí. Cada una de ellas cumple una función específica y presenta una serie de limitaciones y requerimientos. La arquitectura BLE está formada por diferentes protocolos, que son las capas que implementan los diferentes formatos de paquete, encaminamiento, multiplexación y decodificación, que permiten que los datos se envíen de manera efectiva entre dispositivos. Además, establece también unos perfiles, que definen cómo deben usarse los protocolos para lograr objetivos particulares. Estos perfiles pueden ser genéricos o específicos. El conjunto de capas conforma la pila de protocolos que se muestra en la Figura 4. Su estructura se divide principalmente en tres partes bien diferenciadas: controlador, *host* y aplicación. En la parte inferior se encuentra el controlador, formado por la capa física (PHY), la capa de enlace (LL) y la interfaz de control del *host* (HCI). En el lado del *host* se encuentra también la interfaz de control del *host* (HCI), el protocolo de control lógico y adaptación de enlace (L2CAP), el protocolo de gestión de seguridad (SMP), el protocolo de atributo (ATT), el perfil de atributo genérico (GATT) y el perfil de acceso

genérico (GAP). Por último, en la parte superior, se encuentra la capa de aplicación, encargada de gestionar los datos. Representa la capa más alta de la pila del protocolo BLE y es la responsable de mantener la lógica, interfaz de usuario y manejo de datos en función de la aplicación que se vaya a implementar. A continuación, se realiza un análisis más detallado de las diferentes capas pertenecientes a los dos primeros grupos: controlador y *host*.

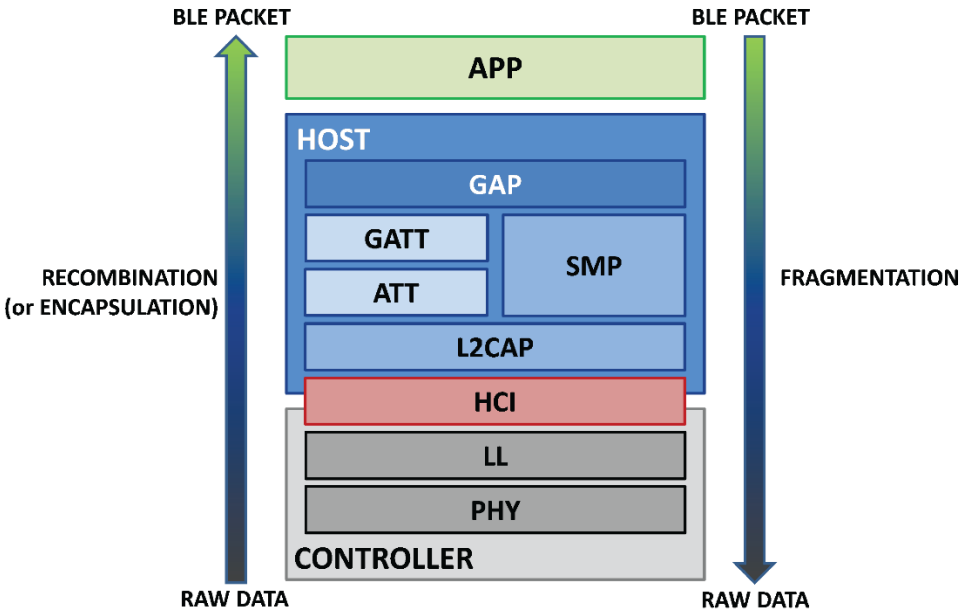

Figura 4. Pila del protocolo BLE [12]

2.2.1 Controlador

La capa más baja de la pila es la capa física, encargada de gestionar de forma directa el circuito de comunicación analógica y realizar funciones de modulación y demodulación de las señales para transformarlas en símbolos digitales. Las señales de *Bluetooth* utilizan la banda de 2.4 GHz ISM para la comunicación, dividida en cuarenta canales de radiofrecuencia espaciados cada 2 MHz [13]. En la Figura 5 se muestra la distribución de estos canales y se señalan los dos tipos de canales RF. El primero de ellos es el de *advertising*, que agrupa tres canales utilizados para encontrar dispositivos, establecer la conexión, y transmisión en modo *broadcast*. El resto de los 37 canales restantes se utilizan para la transmisión bidireccional de datos entre los dispositivos conectados.

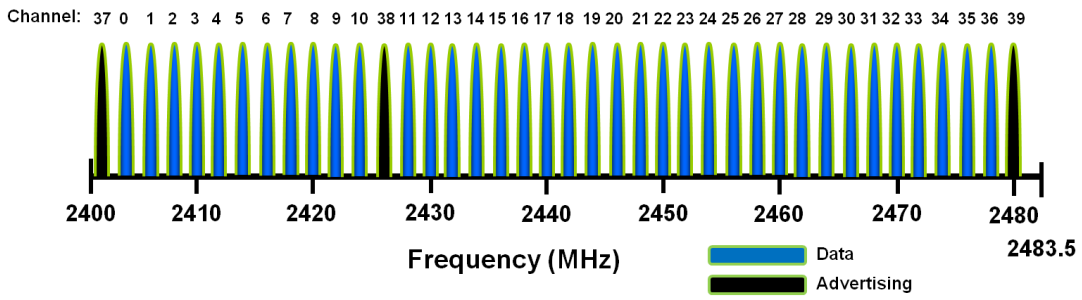

Figura 5. Distribución de canales en el espectro de frecuencias [14]

La capa de enlace se encarga de interactuar directamente con la capa física. Tiene como función principal gestionar cómo un dispositivo se conecta con otros, es decir, determina el estado del enlace *Bluetooth*. Es la capa responsable de los procesos de *scanning* y *advertising.* Además, ofrece la definición de los roles de *Advertiser, Scanner, Master* y *Slave*, que permiten identificar de forma lógica el rol de cada dispositivo en el proceso de comunicación. Su funcionamiento se puede describir como una máquina de estados, que se representa en el esquema de la Figura 6, en la que se definen cinco estados: escaneo, *standby*, inicialización, aviso y conexión. Cuando se establece la conexión entre dos dispositivos, uno de ellos juega el papel de *Master* y el otro es de *Slave*.

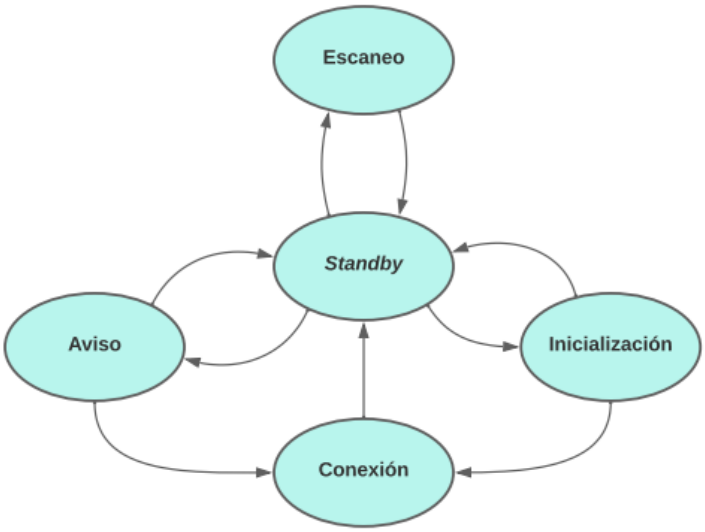

Figura 6. Máquina de estados de la capa de enlace

El estado de *standby*, estado predeterminado en la capa de enlace, no permite la recepción ni transmisión de paquetes. Se puede llegar a este estado desde cualquier otro. En el estado de aviso los dispositivos *peripheral* transmiten paquetes de *advertising*. En este estado también se puede escuchar las respuestas de los dispositivos *central* a estos paquetes. Se llega a él desde el estado *standby* cuando la capa de enlace inicia el proceso de *advertising*. En el estado de escaneo los dispositivos *centrales* escuchan los paquetes de *advertising*. A este estado

se puede llegar desde *standby* cuando la capa de enlace comienza el proceso de *scanning*. En el estado de inicialización se escuchan los paquetes recibidos desde un dispositivo periférico y se inicia una conexión. En general, este es el estado por el que pasa un dispositivo *central* antes de pasar al estado de conexión. Por último, en el estado de conexión el dispositivo *central* está conectado a otro dispositivo *peripheral*, definiéndose los roles de *Master* y *Slave*. Si se llega a este estado desde el estado de inicialización, el dispositivo actúa como *Master*, mientras que, si se llega desde el estado de aviso, actuará como *Slave.*

2.2.2 *Host*

La conexión física entre el *host* y el controlador se realiza gracias a la capa HCI. La especificación define varias interfaces físicas, entre las que se encuentran UART (*Universal Asynchronous Receiver Transmitter*), 3Wire UART, USB (*Universal Serial Bus*) y SDIO (*Security Digital Input Output*) [15].

La siguiente capa de la pila está formada por el protocolo L2CAP. Se trata de un protocolo basado en *Bluetooth* clásico, pero optimizado y simplificado para su uso en BLE. Se encarga de la multiplexación de datos de las capas superiores junto con el control de señal de la capa de enlace. No se utiliza la segmentación de datos, ni mecanismos de retransmisión y control de flujo. Por tanto, es la capa encargada de dar formato a mensajes provenientes de las capas superiores y encapsularlos en paquetes estándar BLE, así como de realizar el proceso inverso. También lleva a cabo el proceso de fragmentación y recombinación de los paquetes que en el nivel de aplicación suponen datagramas de gran cantidad de bytes.

Para BLE, la capa L2CAP es la encargada de dar acceso y soporte a los protocolos ATT y SMP. El protocolo ATT define la comunicación entre dos dispositivos, siguiendo una arquitectura cliente-servidor y permitiendo al dispositivo servidor exponer una serie de atributos y valores al dispositivo cliente. En BLE, cada dispositivo es un cliente, servidor o ambos, independientemente de si es un dispositivo *Master* o *Slave.* Cada servidor contiene datos almacenados en forma de atributos, que tienen asignado un identificador de atributo de 16 bits (para acceder a un valor del atributo), un identificador único universal o UUID (que especifica el tipo y la naturaleza de los datos asociados al valor), un conjunto de permisos y un valor. Para que el cliente pueda acceder y modificar los atributos del servidor, necesita enviar solicitudes a través del controlador. El servidor responderá con el valor del atributo o con un acuse de recibo. En caso de una operación de lectura, el cliente analiza el valor e interpreta el tipo de datos en función de UUID del atributo. Si es una operación de escritura, el cliente proporcionará datos para el tipo del atributo, que podrán ser rechazados por el servidor si no son válidos.

Por otra parte, el protocolo SMP proporciona un *framework* para generar y distribuir claves de seguridad entre dos dispositivos. Su función principal es la de gestionar el intercambio de mensajes en las fases de emparejamiento, proporcionando la capacidad de generar e intercambiar claves de seguridad a la pila de protocolos. Esto supone un mecanismo de seguridad que permite al dispositivo utilizar y cambiar con frecuencia sus direcciones privadas.

Dentro del bloque del *host* se utilizan los dos perfiles genéricos definidos en BLE. El primero es GATT, que describe la estructura de los servicios, es decir, define cómo dos dispositivos BLE transfieren información. También permite que se produzca el intercambio correspondiente de características entre los dispositivos. Por tanto, define un modelo básico de datos y procedimientos para permitir a los dispositivos leer, escribir, enviar o descubrir elementos de datos entre ellos. En este perfil se define cómo se organizan y se intercambian los datos entre las aplicaciones. Los datos están encapsulados en servicios que constan de una o más características. Cada característica es la unión de una parte de datos del usuario junto con otra parte de metadatos, que se corresponde con información descriptiva sobre ese valor, como sus propiedades, su nombre visible o unidades, entre otros. El perfil de atributo genérico define dos tipos de roles que pueden adoptar los dispositivos BLE: cliente y servidor. El cliente envía solicitudes a un servidor y recibe respuestas. Inicialmente, no conoce los atributos del servidor, por lo que debe realizar consultas para completar el descubrimiento de servicios. Una vez determinada la presencia de servicios, puede comenzar a leer y escribir los atributos encontrados en el servidor, así como recibir actualizaciones. En cambio, el servidor recibe las solicitudes de un cliente y devuelve respuestas. También envía las actualizaciones y es el rol responsable de almacenar y poner a disposición del cliente los datos organizados en atributos. Cada dispositivo BLE debe disponer de al menos un servidor GATT que sea capaz de responder a las solicitudes de los clientes.

El segundo se corresponde con el perfil GAP, que permite que un dispositivo sea visible hacia el resto de los dispositivos y determina cómo puede interactuar un dispositivo con otro. Establece distintas normas y conceptos para estandarizar las operaciones de más bajo nivel como los roles de interacción, modos de operación y transición entre ellos, procedimientos para el establecimiento de la comunicación o modos de seguridad y procedimientos. Interviene en la definición de características de los roles que se le asignan a un dispositivo. Los roles que pueden definirse se engloban en cuatro grupos principalmente: *broadcaster, observer*, *central* y *peripheral* [13]. El rol *broadcaster* permite la transmisión de datos por los canales de aviso y no permite la conexión con otros dispositivos. Se envían periódicamente paquetes de *advertising* con datos que estarán accesibles para cualquier dispositivo que esté escuchando. El rol

broadcaster utiliza el rol de *advertiser* de la capa de enlace. El dispositivo *observer* se encarga de recibir o escuchar los datos transmitidos por el *broadcaster*. Este rol utiliza el rol de *scanner* de la capa de enlace. El dispositivo *central*, asociado al rol *Master* de la capa de enlace, es el encargado de controlar las conexiones. Es el dispositivo capaz de establecer múltiples conexiones con otros dispositivos y es siempre el iniciador de las conexiones, permitiendo que el resto de los dispositivos entren en la red. El rol de *peripheral* se asigna a los dispositivos que utilizan una sola conexión, por lo que está asociado con el rol de *Slave* de la capa de enlace. Emplea paquetes de *advertising* para permitir ser encontrado por algún dispositivo *central* y poder establecer una conexión posterior. Cada dispositivo puede operar en uno o más roles a la vez, ya que la especificación BLE no impone restricciones en este aspecto.

2.3 Topología de red

La topología de red de BLE es de tipo estrella. Los dispositivos centrales o *Master* pueden tener varias conexiones de capa de enlace con periféricos o dispositivos *Slave,* y simultáneamente realizar búsquedas de otros dispositivos. Sin embargo, un dispositivo periférico solo puede tener una conexión de capa de enlace con un único *Master*. Estos dispositivos BLE tienen dos formas de comunicación, a través de *Broadcasting* o mediante conexiones sujetas a las pautas establecidas por el perfil GAP.

Gracias a la comunicación *Broadcasting* se pueden enviar datos a cualquier dispositivo receptor o en proceso de *scanning*, que se encuentre dentro del rango de alcance o escucha. En la Figura 7 se muestra la comunicación unidireccional entre un dispositivo central, o *broadcaster*, con una serie de dispositivos que adoptan el rol de *observer*, únicamente recibiendo los datos que se envían. El dispositivo *bradcaster* envía paquetes de *advertising* de forma periódica a cualquier dispositivo que esté dispuesto a recibirlos, por lo que es posible realizar el envío a más de un dispositivo BLE. Los paquetes de *advertising* estándar tiene un *payload* de 31 bytes que, en caso de no ser suficiente, puede acompañarse de uno secundario de otros 31 bytes. Es un método rápido y fácil de usar pero que no presenta garantías en términos de seguridad, por lo que no es aconsejable para información sensible.

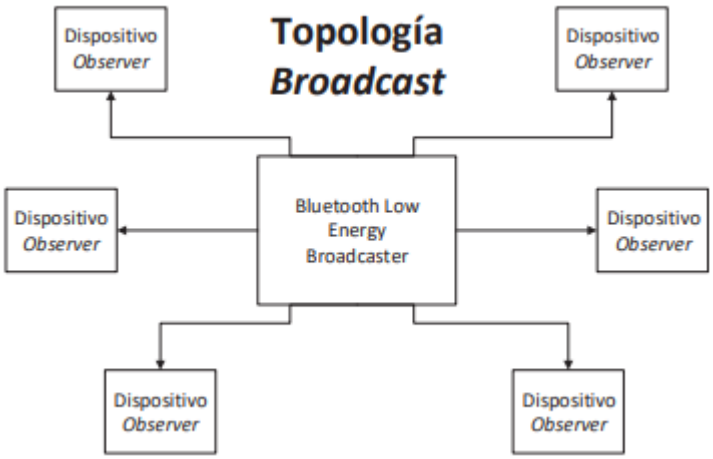

Figura 7. Topología Broadcasting BLE [16]

Cuando se pretende realizar una comunicación bidireccional, o bien se necesitan más bytes de los que se pueden enviar en los paquetes de *advertising,* será necesario establecer una conexión entre los dispositivos. La conexión define el intercambio de paquetes de forma periódica, permanente y privada. En la conexión intervienen los roles de *Master* y *Slave.* El dispositivo *central* o *Master* busca paquetes de *advertising* para poder iniciar la conexión. Una vez establecida, controla la temporización e inicia el intercambio periódico de datos. Por otro lado, el dispositivo *peripheral* o *Slave* envía paquetes de *advertising* para la conexión, y acepta conexiones entrantes.

En la Figura 8 se muestra cómo el dispositivo *central* está capacitado para recibir paquetes de *advertising* de los dispositivos *peripheral*, y puede enviar solicitudes para el establecimiento de conexión exclusiva. Cuando se establece la conexión, el dispositivo periférico abandona el proceso de *advertising* y se produce el intercambio de datos.

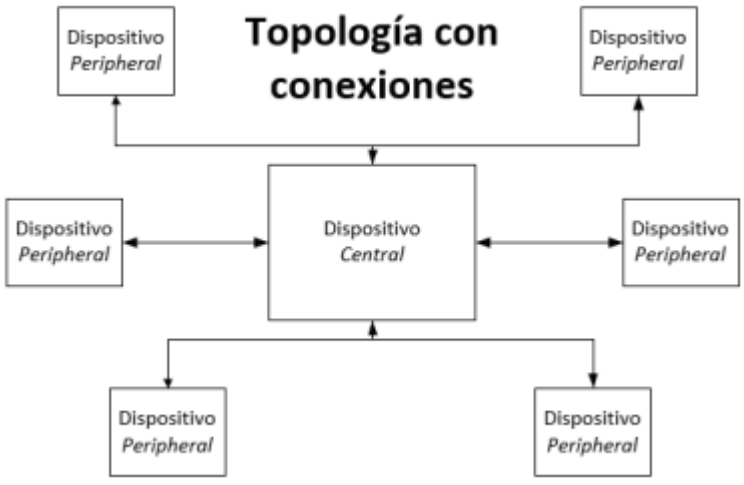

Figura 8. Topología mediante conexiones BLE

Con la topología de conexiones se consigue utilizar mucha menos energía que con el modo *Bradcast*, ya que los datos pueden enviarse solo cuando hay nuevos valores disponibles, en lugar de enviar paquetes de *advertising* sin saber quién está escuchando y con qué frecuencia. También, es conveniente destacar que ambas topologías pueden combinarse formando una topología mixta en una red BLE.

2.4 Atributos, Servicios, Características y Notificaciones

La jerarquía que todo dispositivo BLE aplica está basada en atributos, servicios, características y descriptores. En un servidor GATT, los atributos se agrupan en servicios. Estos servicios pueden están formados por características y estas, a su vez, pueden incluir descriptores. Esta organización jerárquica se puede ver representada en la Figura 9. Los servicios y características se identifican en BLE mediante UUID, que son identificadores únicos universales de 16 bytes. Para no tomar gran parte de los bytes de datos de la capa de enlace y, por tanto, una mayor eficiencia, la especificación BLE agrega dos formatos UUID adicionales, uno de 16 y otro de 32 bits, que solo pueden ser utilizados en la especificación *Bluetooth*.

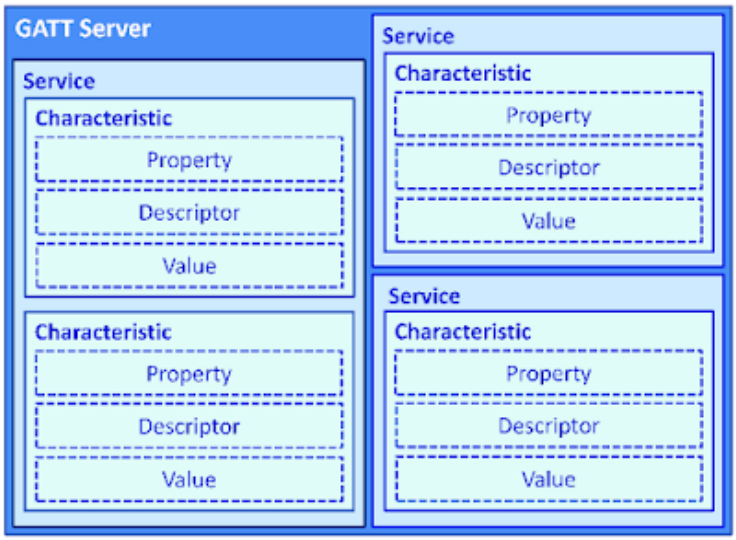

Figura 9. Jerarquía de servicios y características de GATT [17]

Por tanto, es conveniente, en primer lugar, introducir el concepto de atributo. Los atributos son bloques de información que constituyen la entidad de datos más pequeña. Pueden contener datos de usuario o metadatos y están siempre ubicados en el servidor. En la Figura 10, se observan los diferentes campos que contienen los datos referentes a los atributos. Presentan un campo de 16 bits a modo de identificador que hace que sea direccionable, un campo de tipo que contiene el UUID y que determina el tipo de datos presentes en el valor del atributo, un campo de permisos que especifica qué operaciones se pueden ejecutar y con qué requisitos de

seguridad (permisos de acceso, cifrado o autorización) y un campo de valor con una longitud máxima de 512 bytes que contiene los datos actuales asociados al atributo y que se podrá leer o escribir por los clientes.

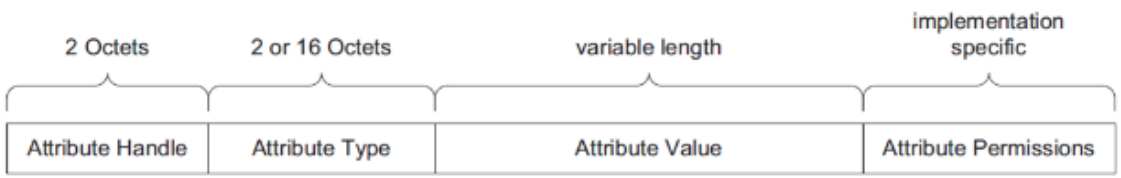

Figura 10. Estructura de un atributo [17]

El protocolo ATT se encarga de almacenar los servicios, características y datos relacionados en una tabla. Los servicios se utilizan para dividir datos en entidades lógicas y contienen secciones específicas de datos, llamados características.

Por otra parte, el concepto de nivel más bajo entre las transacciones GATT son las características, que encapsulan un único tipo de dato. Existe libertad para usar tanto las características estándar oficialmente definidas por *Bluetooth*, como características propias personalizadas para un periférico y aplicación determinadas. Las características son los elementos principales que permiten la interacción con los dispositivos periféricos BLE. Pueden ser de escritura o de lectura, de manera que pueden utilizarse para realizar comunicaciones bidireccionales de forma sencilla.

Para llevar a cabo la lectura de las características de una forma eficiente es necesario realizar suscripciones a notificaciones. Sin el uso de las notificaciones, cuando se captura algún dato concreto, se debe ir leyendo progresivamente el valor de la característica correspondiente asociada al mismo para obtenerlo. Sin embargo, en muchas ocasiones interesa que sea el servidor el que envíe el valor requerido hacia el cliente en un momento determinado. Este proceso es algo parecido a la suscripción a un servicio de mensajería, en el que los mensajes llegan directamente sin una intervención manual ni una solicitud. Es un sistema muy cómodo para recibir datos de sensores. Por ejemplo, si se cuenta con un sensor de temperatura, en lugar de realizar una llamada cada cierto tiempo para comprobar si hay nuevas lecturas (siendo la temperatura un parámetro que no sufre modificaciones de una forma rápida) se puede realizar una suscripción a la notificación de su característica, consiguiendo así que el servidor envíe el dato cuando exista una nueva lectura. De esta forma se consigue eliminar la necesidad de comprobar periódicamente si los datos han sido modificados, obteniendo un ahorro de energía.

2.5 Paquetes BLE

Los paquetes del protocolo BLE siguen una misma estructura, independientemente de si son paquetes de *advertising* o de datos. Estos paquetes son más pequeños, en comparación con *Bluetooth* clásico, con el fin de minimizar el tiempo de activación de la radio para la transmisión y recepción y, en consecuencia, el consumo de energía. Además, las transacciones en BLE, como la mayoría de las transacciones en redes de sensores inalámbricos, solo requieren una pequeña cantidad de intercambio de datos. En la Figura 11, se muestra un esquema del formato de un paquete BLE, donde se diferencian cuatro campos.

Figura 11. Formato del paquete BLE

El primero de ellos es un campo de preámbulo de 8 bits que se utiliza para la sincronización y para identificar si los paquetes se envían desde canales de *advertising* o canales de datos. El segundo bloque del paquete se corresponde con la dirección de acceso, que ocupa 32 bits y tiene diferentes valores según el tipo de canal. A continuación, se encuentra la unidad de datos de protocolo (PDU), de un tamaño variable entre 2 y 39 bytes [18]. La estructura del PDU es diferente en función del canal de destino. En la Figura 12, se muestran las diferencias existentes entre un paquete para el canal de *advertising* (a) o para el canal de datos (b). Se observa que, para el canal de *advertising*, la PDU se divide en dos campos. El primero es una cabecera de 2 bytes y el segundo un campo variable de carga útil de entre 0 y 37 bytes. En cambio, para los canales de datos, el campo de PDU se compone de tres partes. En primer lugar, una cabecera de 2 bytes, una carga útil de hasta 255 bytes, y un campo de verificación MIC (*Message Integrity Check*) de 4 bytes. MIC evita ataques en paquetes cifrados, agregando algunos bytes para que los paquetes se encuentren seguros ante ataques o manipulaciones de cambio de bit [19]. Durante estos ataques, un intruso intercepta el mensaje cifrado, alterándolo y retransmitiéndolo, haciendo que el receptor acepte el mensaje retransmitido como correcto.

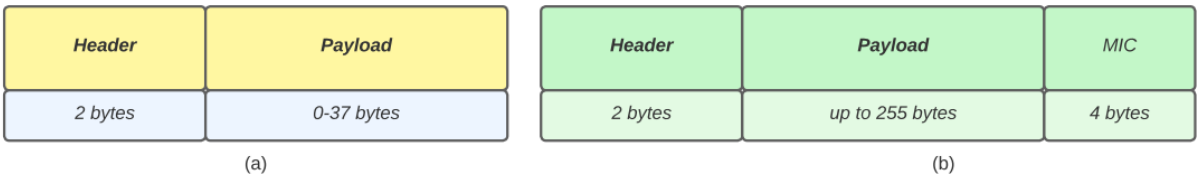

Figura 12. PDU para el canal de advertising (a) y de datos (b)

2.6 *Advertising* y *Scanning* en BLE

Para la obtención de datos o el establecimiento de conexión se requieren los métodos de *scanning* y *advertising*. Cada paquete de *advertising* se envía por el dispositivo de tipo *advertiser* sin conocer previamente si existe algún dispositivo que pueda recibir las señales. Estos paquetes serán enviados a una cierta velocidad definida por el intervalo de *advertising*, de hasta 20 ms. Si el intervalo es menor, se enviarán paquetes a mayor frecuencia, consiguiendo una mayor probabilidad de detección por parte de un dispositivo *scanner*, pero con un incremento en el costo de consumo de energía. Como se comentó anteriormente, el envío de paquetes de *advertising* se realiza utilizando tres canales de frecuencia definida. Como los dispositivos *advertiser* y *scanner* no han sido previamente sincronizados, la detección se realizará de forma aleatoria cuando se superpongan las frecuencias de los procesos de *advertising* y *scanning*, tal y como se muestra en la Figura 13.

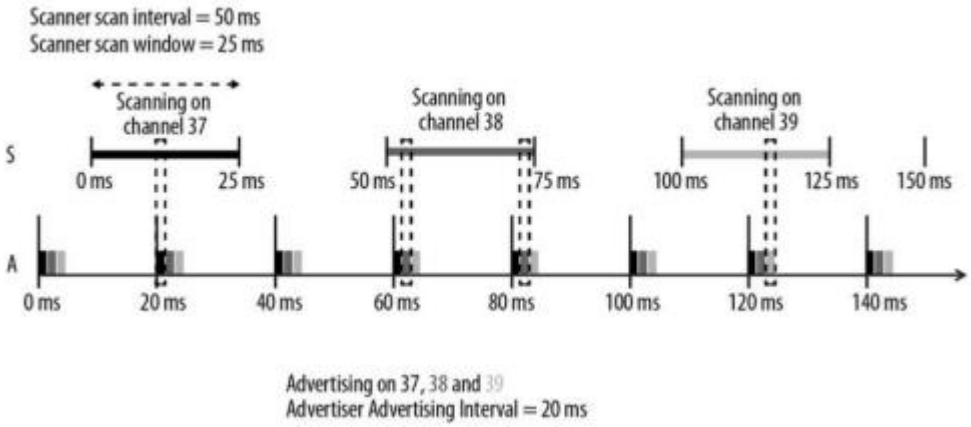

Figura 13. Procesos de scanning y advertising [11]

Los parámetros *scanner scan interval* y *scanner scan window* definen con qué frecuencia y por cuánto tiempo va a estar explorando un dispositivo *scanner* los paquetes de *advertising*, respectivamente. BLE define dos tipos de procedimientos de *scanning*. El primero de ellos pasivo, donde el dispositivo *scanner* escucha los paquetes de *advertising*, pero el dispositivo *advertiser* no conoce si los paquetes han sido recibidos, y el procedimiento activo, en el que el

dispositivo *scanner* envía un paquete de solicitud de *scanning* tras recibir un paquete de *advertising*. Cuando el dispositivo *advertiser* lo recibe, responde también con un paquete de *scanning*. En la Figura 14, se representa la diferencia entre los dos tipos de procedimientos de *scanning*.

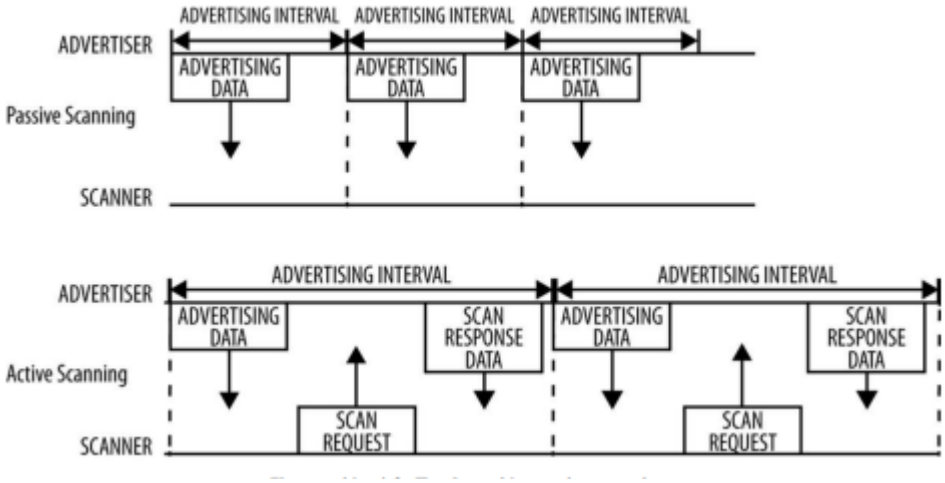

Figura 14. Tipos de procedimientos de scanning [11]

CAPÍTULO 3 Proceso de clasificación

En este capítulo se presenta una breve introducción a *Machine Learning* y los diferentes métodos de clasificación. Sin embargo, se hará especial hincapié en detallar el algoritmo de clasificación *Support Vector Machine* (SVM), al ser el método elegido para el desarrollo de la plataforma *hardware/software* de este TFG. Para completar todos los aspectos relacionados con el proceso de clasificación se incluye también una presentación de la herramienta *software* LIBSVM que proporciona una librería que fundamenta alguno de los procedimientos necesarios para llevar a cabo el proceso de reconocimiento de los gestos.

3.1 Introducción

En la actualidad, las aplicaciones que realizan tareas con el tratamiento de la información deben tener en cuenta que la cantidades y dimensión de los datos son factores que están constantemente en crecimiento. Este aspecto dificulta la identificación de relaciones o el establecimiento de patrones, lo que conlleva la necesidad de utilizar computadoras que realicen cálculos matemáticos complejos y que permitan entender los que los datos reflejan para extraer conclusiones certeras. Bajo esta premisa, aparece el concepto de *Machine Learning*, que es una disciplina dentro del campo de la inteligencia artificial centrada en el entrenamiento de modelos matemáticos a través de datos, con el fin de que algoritmos que incorporen estos modelos sean capaces de realizar tareas inteligentes, generalmente basadas en el reconocimiento de patrones multiparamétricos [20]. La técnica de *Machine Learning* pretende crear sistemas automáticos capaces de modificar su comportamiento en base a unos datos de entrada y descubrir patrones imperceptibles para los humanos por la complejidad de los datos.

3.1.1 Tipos de aprendizaje

Se pueden definir tres grandes grupos de algoritmos de *Machine Learning*, en función de las preguntas a las que responden y del tipo de información disponible para su entrenamiento. El primero de ellos es el de algoritmos de aprendizaje no supervisado, que únicamente tienen en cuenta los datos de entrada y no disponen de etiquetas para clasificarlos, por lo que no se tienen resultados conocidos. En este caso, el algoritmo debe deducir y encontrar patrones, en base a los datos de entrada a través de métodos matemáticos, que organicen la información por su similitud. Los algoritmos de aprendizaje no supervisado más representativos son el de *Clustering* y el de Asociación. El primero descubre los grupos naturalmente presentes en los datos, mientras que el segundo descubre las reglas que los propios datos describen.

El segundo grupo de algoritmos de *Machine Learning* es el de aprendizaje supervisado. En este caso, se cuenta con una serie de variables de entrada, o predictores, que se utilizan para obtener una variable de salida cuyos posibles valores son conocidos. Por tanto, los datos de entrenamiento son previamente etiquetados o preclasificados, de modo que el algoritmo cuenta con las preguntas o características y las respuestas o etiquetas. Este algoritmo se centra en dos tipos de problema o entrenamiento, el de clasificación y el de regresión [21]. Para la clasificación, el algoritmo encuentra patrones en los datos de entrada y los clasifica en grupos. Posteriormente, compara los nuevos datos y los ubica en uno de los grupos. Por su parte, para la regresión, el resultado esperado es un número, ya que este método no ubica los datos en grupos, sino que devuelve un valor específico.

Para comprender de una forma más clara las diferencias entre ambos algoritmos se hace referencia a la Figura 15, donde se observa la separación de las muestras que realiza cada uno de ellos. En la gráfica de la izquierda, que se corresponde con el algoritmo de aprendizaje supervisado, el espacio bidimensional se divide en dos subespacios mediante un modelo lineal, de forma que los casos que se encuentran a un lado son controles y en el lado contrario se sitúan los casos. Además, cada clase se etiqueta con su clase correcta. A la derecha, con el método de aprendizaje no supervisado, el algoritmo es capaz de rodear subespacios, cada uno perteneciente a un grupo de muestras que se parecen entre sí y que, a su vez, se distinguen del otro grupo.

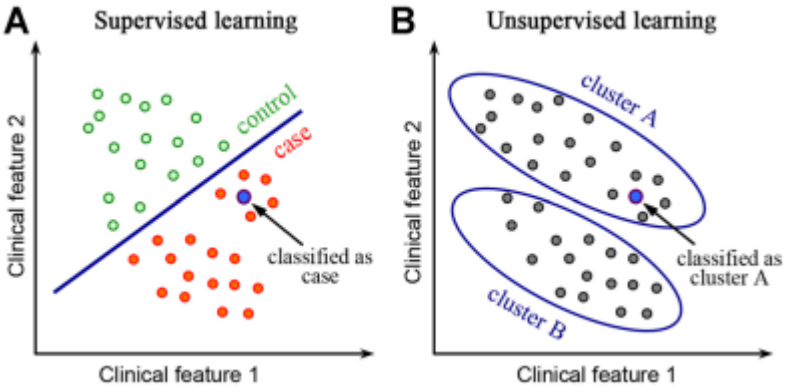

Figura 15. Algoritmo de aprendizaje supervisado y no supervisado [20]

Por último, existe un tercer grupo de algoritmos que combina tanto datos etiquetados como no etiquetados para generar el clasificador. Estos algoritmos se denominan semi-supervisados y deben aprender las estructuras para organizar los datos y realizar también predicciones.

3.1.2 Modelos de *Machine Learning*

Los algoritmos de *Machine Learning* se pueden agrupar en tres modelos principales: modelos lineales, modelos de árbol, y redes neuronales. Los modelos lineales tratan de encontrar una línea que se ajuste a la nube de puntos que disponen. Estos modelos pueden presentar el problema de *overfit*, ajustándose demasiado a los datos disponibles y no consiguiendo resultados óptimos para los nuevos datos. Son modelos relativamente simples que no se aconsejan para comportamientos complejos.

Los modelos de árbol son modelos precisos y estables que basan su comportamiento en la construcción de reglas de decisión en forma de árbol, como se muestra en la Figura 16. El nodo raíz, identificado en color verde, representa a toda la población o muestra. En color azul se identifica un nodo de decisión, que determina cuándo se divide un subnodo en subnodos adicionales. El resto de los nodos, identificados en color rosa, son nodos terminales. Cada una de las subsecciones del árbol se denomina rama. Pueden representar relaciones no lineales para resolver problemas y, al ser modelos más elaborados, requieren de un mayor rendimiento. La idea principal de los árboles de decisión es dividir las observaciones del nodo raíz en conjuntos homogéneos mediante reglas binarias a partir de variables [22].

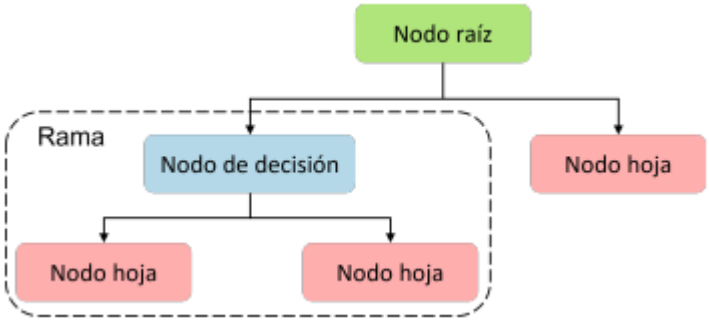

Figura 16. Estructura de un árbol de decisión [22]

Por último, las redes neuronales son los modelos que pretenden replicar el comportamiento de las neuronas en los cerebros biológicos. Requieren un lento proceso de aprendizaje y requieren de mucha capacidad de cómputo. Logran establecer relaciones no lineales entre las variables de entrada y las variables de salida. Una de las principales ventajas de este modelo es que permite analizar, clasificar y procesar la información en patrones complejos. La conexión entre los diferentes nodos que conforman la red neuronal se realiza a través de sinapsis, y a cada nodo se le da diferente peso dependiendo del impacto que tenga en la predicción que se desea realizar [23].

3.2 Algoritmo *Support Vector Machine*

SVM (*Support Vector Machine*) es un método de aprendizaje automático muy reconocido para la clasificación, regresión y otras tareas de aprendizaje. A partir de un conjunto de datos de entrenamiento dado, el clasificador etiqueta las clases y construye un modelo que predice la clase que proviene de nuevas muestras. Este método ha conseguido una gran popularidad por su capacidad para producir buenos modelos en diversos tipos de aplicaciones. En un principio, fue diseñado para resolver problemas de clasificación binaria, pero posteriormente su uso se extendió a tareas de regresión, clasificación multicategorías, agrupamiento, etc. SVM pretende construir modelos confiables, consiguiendo predicciones certeras a costa de producir ciertos errores.

Este modelo de clasificación representa los puntos de la muestra en el espacio, separando las clases en espacios lo más amplios posibles mediante un hiperplano de separación [24]. Los vectores de soporte son aquellos puntos que definen los márgenes de separación entre las clases y el hiperplano. Con la Figura 17 se pretende aclarar de forma gráfica los aspectos introducidos. Se diferencian dos observaciones, una en color azul y otra en color rosa, correspondientes a dos clases distintas. Entre ellas, el hiperplano de margen máximo se ha representado con una línea continua. El margen está determinado por el espacio entre el hiperplano y cualquiera de las líneas discontinuas y se indica mediante flechas. Los puntos que se encuentran sobre las líneas discontinuas constituyen los vectores de soporte.

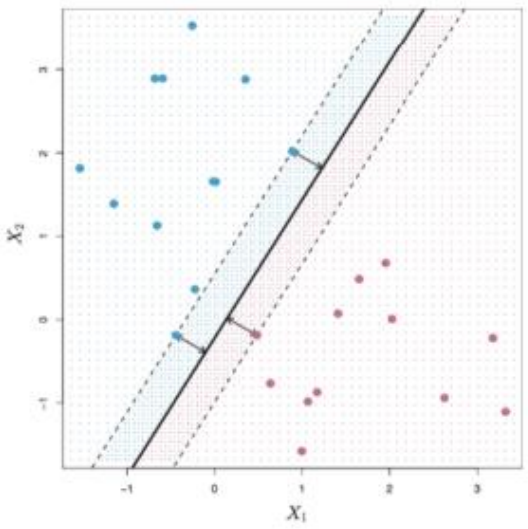

Figura 17. Máquina de Soporte Vectorial Lineal [24]

Por tanto, el hiperplano se muestra como una línea continua y los márgenes se muestran como líneas punteadas. El clasificador SVM se ajusta a un pequeño conjunto de datos de entre

todos los disponibles. En la gráfica de la izquierda de la Figura 18 se comprueba cómo las observaciones 2 (de color rosa) y 9 (de color azul) están en el margen. Las observaciones 3, 4, 5, 6 (de color rosa) y las observaciones 7 y 10 (de color azul) se encuentran en el lado correcto del margen, mientras que las observaciones 1 (de color rosa) y 8 (de color azul) se encuentran en el lado incorrecto del margen. En este caso no hay observaciones en el lado incorrecto del hiperplano. Sin embargo, sí se observa ahora que en la gráfica de la derecha aparecen dos puntos adicionales. Las observaciones 11 y 12 están en el lado incorrecto del hiperplano y en el lado incorrecto del margen.

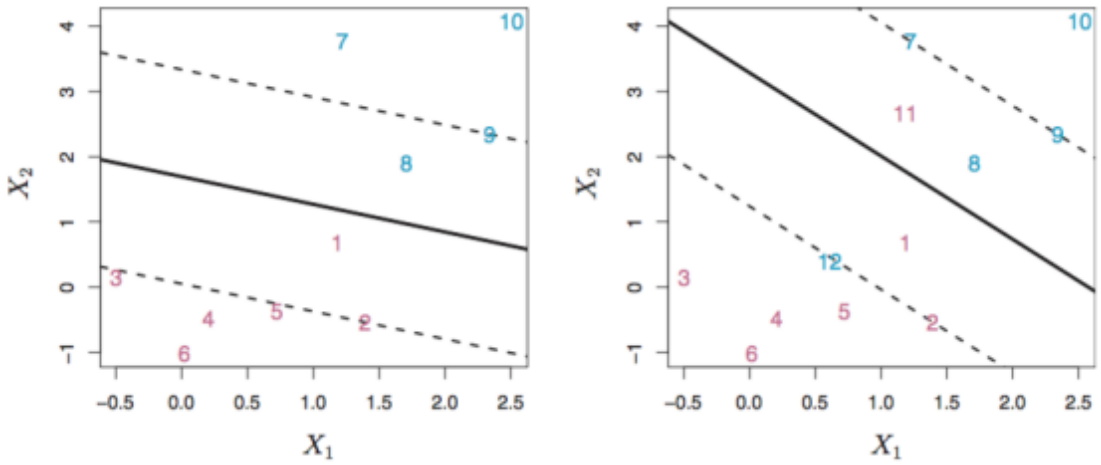

Figura 18. Observaciones en hiperplanos de SVM [24]

Para poder dotar al clasificador de cierta flexibilidad, SVM posee un parámetro ajustable denominado C. Este parámetro controla la compresión entre los márgenes y los errores de entrenamiento. De esta forma se consigue un margen adaptable que permite ciertos errores en la clasificación en función de su valor. Por ejemplo, en la etapa de entrenamiento, un valor elevado del parámetro C supone un mayor número de valores dentro del lado incorrecto del margen. Este comportamiento se puede ver reflejado en las dos gráficas de la Figura 19. En la gráfica de la izquierda se tiene un valor mayor para el parámetro C y, por tanto, un mayor margen. A medida que C disminuye, el margen se estrecha, tal y como se observa en la gráfica de la derecha, y la tolerancia para que las observaciones estén en el lado incorrecto del margen disminuye.

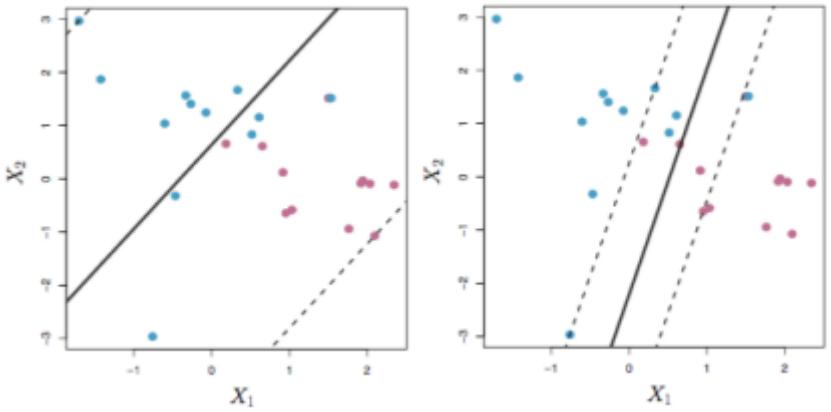

Figura 19. Comportamiento de SVM para distintos valores del parámetro C [24]

Hasta ahora, en los ejemplos mostrados, solo se han representado espacios lineales. Sin embargo, no siempre el conjunto de entrada es linealmente separable y en la mayoría de los casos se requiere de una transformación de espacios. Esto ocurre cuando SVM trata con más de dos variables predictoras o clases, en las que no se puede utilizar una línea recta para realizar la separación. En estos casos se utiliza la función *kernel*, encargada de mapear el espacio de entradas a un nuevo espacio de características de mayor dimensión, tal y como se muestra en la Figura 20. Tras la transformación, el límite entre las dos categorías se puede definir por un hiperplano de una forma más sencilla.

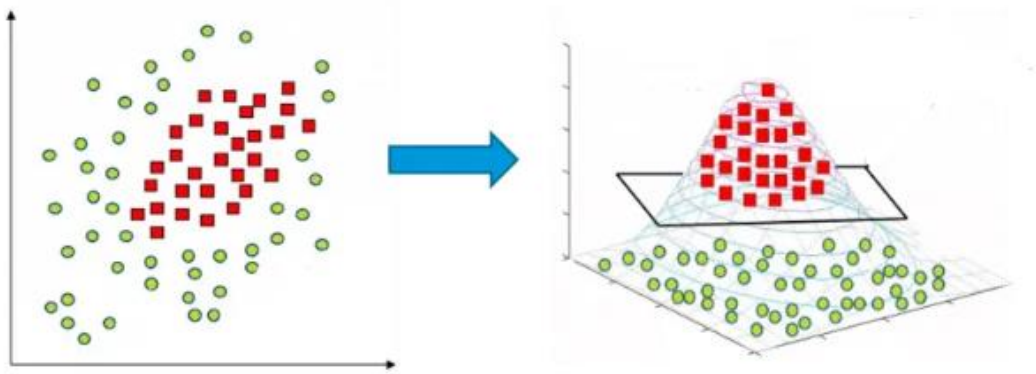

Figura 20. Aplicación kernel en SVM [25]

Por tanto, se crea una nueva dimensión en la que es posible encontrar un hiperplano para separar las clases con una superficie de decisión. El algoritmo dependerá de los datos de entrada para un conjunto de datos de entrenamiento y se buscará encontrar la máxima separación entre clases, con el fin de realizar detecciones más certeras. De esta forma, se pretende conseguir separar las clases después de realizar esta transformación, simplificando potencialmente los límites de decisión complejos no lineales para hacerlos lineales en el espacio

dimensional de características. En este proceso, los datos no se tienen que transformar explícitamente, lo que supondría una alta carga computacional.

La función *kernel*, por tanto, basa su funcionalidad en la representación de la información y es la encargada de definir los límites de decisión entre las clases. Las representaciones *kernel* ofrecen una solución alternativa al proyectar los datos en un espacio de características de alta dimensión para aumentar la potencia de cálculo de las máquinas de aprendizaje lineal [26]. Las funciones *kernel* más habituales son *kernel* lineal, polinómica, radial, gaussiana o sigmoidal. Sin embargo, también puede emplearse una combinación de funciones *kernel*.

Es conveniente profundizar en el *kernel* RBF (*Radial Basic Function*), ya que será el *kernel* utilizado en la plataforma desarrollada. Este tipo de *kernel* suele utilizarse cuando el conjunto de datos no es lineal y los límites presentan forma de curva. Utiliza dos parámetros principales, *gamma* y C, relacionados con la extensión de la región de decisión y la penalización por la clasificación errónea de un punto de datos, respectivamente. El parámetro *gamma* define hasta dónde llega la influencia de un solo ejemplo del entrenamiento y el parámetro C compensa la clasificación correcta de los ejemplos de entrenamiento con la maximización del margen de la función de decisión, comportándose como un parámetro de regularización en SVM. El comportamiento del modelo es muy sensible al parámetro *gamma*, ya que si su valor es muy pequeño el modelo estará demasiado restringido y no podrá capturar la complejidad de los datos.

Para la construcción de modelos de clasificación se busca la utilización del menor número de atributos posibles para conseguir resultados exitosos [27]. Por eso, el proceso de elección del tipo y el número de atributos se considera uno de los factores claves al determinar la efectividad del modelo de discriminación construido.

3.3 Biblioteca LIBSVM

LIBSVM (*Library for Support Vector Machines*) es actualmente uno de los entornos de SVM más utilizados. La herramienta LIBSVM es un *software* simple, fácil de usar y eficiente para la clasificación y regresión SVM. Resuelve la clasificación C-SVM, la clasificación nu-SVM, la regresión épsilon-SVM y la regresión nu-SVM [28]. Además, proporciona una herramienta de selección automática de modelos para la clasificación C-SVM y soporta clasificación multiclase. Es una herramienta de aprendizaje común y ampliamente utilizada debido a su alta precisión en tareas de clasificación. Esta biblioteca SVM viene desarrollándose desde el año 2000 por Chih-

Chung Chang y Chih-Jen Lin con el objetivo de ayudar a los usuarios a aplicar fácilmente SVM a sus aplicaciones [29].

Por lo general, la utilización del *software* LIBSVM se divide en dos pasos. En primer lugar, se necesita elaborar un modelo a partir del entrenamiento con un conjunto determinado de datos. A continuación, podrá utilizarse el modelo generado para predecir la información de un conjunto de datos de prueba. LIBSVM soporta principalmente tres tareas de aprendizaje: SVC (*Support Vector Classification*) cuando se trabaja con dos o más clases, SVR (*Support Vector Regression*) y *One-class* SVM cuando se utiliza una única clase. Para los dos primeros casos, LIBSVM es también capaz de generar estimaciones de probabilidad.

Para el desarrollo de este TFG, será necesario tener en cuenta la diferenciación entre más de una clase asociada a cada uno de los gestos que se desea identificar, por lo que se utilizará un clasificador SVC. Con esta clasificación se pretende conseguir mayor robustez a observaciones individuales y una mejor clasificación de la mayoría de las observaciones de entrenamiento y prueba. Mediante la clasificación multiclase SVC se obtienen los vectores de soporte y los parámetros del *kernel* en el modelo de predicción. Para la clasificación multiclase, LIBSVM implementa el método *one-against-one*. A partir de una clase dada, este método estudia todos los posibles pares de clases, entre la dada y cada una de las restantes, induciendo un clasificador para cada uno de ellos. Por tanto, partiendo de un número total de clases, denominado *n*, se generan $\frac{n \times (n-1)}{2}$ clasificadores. Cada uno realiza el proceso de entrenamiento de los datos de dos clases. Para llevar a cabo la clasificación, se emplea una estrategia de votación. En cada clasificación binaria se emite un voto para cada muestra. De esta forma, a cada muestra se le asigna la clase que haya recibido un número mayor de votos. Si se consigue un empate en el número de votos entre dos clases, se elige la clase que aparezca en primer lugar en la matriz de almacenamiento de nombre de las clases. El número de clasificadores creados con este método es alto, sin embargo, consigue obtener un entrenamiento más rápido.

El paquete LIBSVM se estructura principalmente en tres directorios diferenciados. El directorio principal contiene el código de los programas centrales en lenguaje C/C++ y datos de muestra. Se encuentra también el archivo *svm.cpp*, que implementa los algoritmos de entrenamiento y prueba. Por otra parte, en el subdirectorio de herramientas se incluyen las herramientas que permiten comprobar el formato de los datos y seleccionar los parámetros de SVM. Por último, se incluyen otros subdirectorios que albergan archivos binarios precompilados, e interfaces para otros lenguajes/*software*.

Todos los algoritmos de entrenamiento y prueba de LIBSVM se implementan en el archivo *svm.cpp*. Las dos subrutinas principales son *svm_train* y *svm_predict*. De la misma manera, dentro de LIBSVM se encuentran los ficheros *svm-train.c*, *svm-scale.c* y *svm-predict.c*. El primero de ellos, *sv-train.c*, es el encargado de recoger los parámetros *gamma* y C, ajustándolos e invocando a la subrutina *svm_train* para generar el modelo de clasificación. La funcionalidad de predicción será la que se integrará en la plataforma *hardware/software* desarrollada en este TFG. Esta funcionalidad se analizará en detalle posteriormente, a partir de la Figura 21, donde se representa el esquema de llamadas de funciones del fichero *svm-predict.c*.

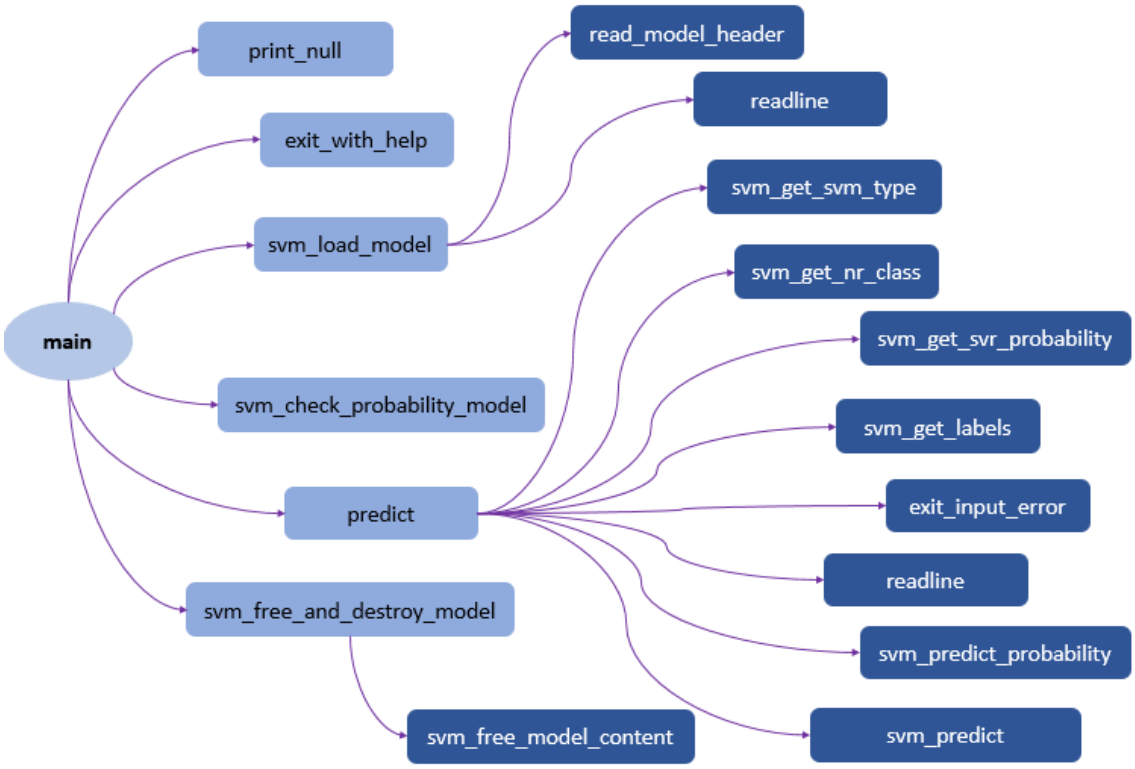

Figura 21. Esquema de funciones del archivo svm-predict.c

Para comprender de forma óptima el método de clasificación es necesario destacar la función *predict*, así como las funciones que derivan de esta. Para ello, se ha recogido una breve descripción de las funciones más relevantes en la Tabla 1.

Tabla 1. Funciones relevantes en el fichero svm-predict.c

Función	Descripción
svm_get_svm_type()	Función que determina el parámetro *svm_type* del modelo. Sus posibles valores vienen definidos en el fichero *svm.h*, tal y como se muestra en la Figura 22 y se relacionan con el tipo de clasificación (C-SVM, nu-SVM…).

svm_get_nr_class()	Función que proporciona el número de clases para un modelo de clasificación.
svm_get_labels()	Función que genera el nombre de etiquetas en una matriz para un modelo de clasificación.
svm_predict_probability()	Función encargada del proceso de clasificación o regresión sobre un vector de prueba a partir de un modelo con información de probabilidad. Para un modelo de clasificación con información de probabilidad esta función da *nr_class* estimaciones de probabilidad en la matriz *prob_estimates*. Devuelve la clase con mayor probabilidad.

```
/* svm_type */
enum { C_SVC, NU_SVC, ONE_CLASS, EPSILON_SVR, NU_SVR };
```

Figura 22. Posibles valores del parámetro svm_type

De la misma manera, en la Figura 23 se muestra el esquema de dependencia de funciones que derivan de la función *svm_predict_probability()*. Entre todas ellas, se puede destacar la función *svm_predict()*, que es la encargada de llevar a cabo el proceso de clasificación o regresión sobre un vector de prueba, devolviendo la clase predicha utilizando un modelo de clasificación. Para un modelo de clasificación, devuelve la clase predicha para el vector de prueba correspondiente.

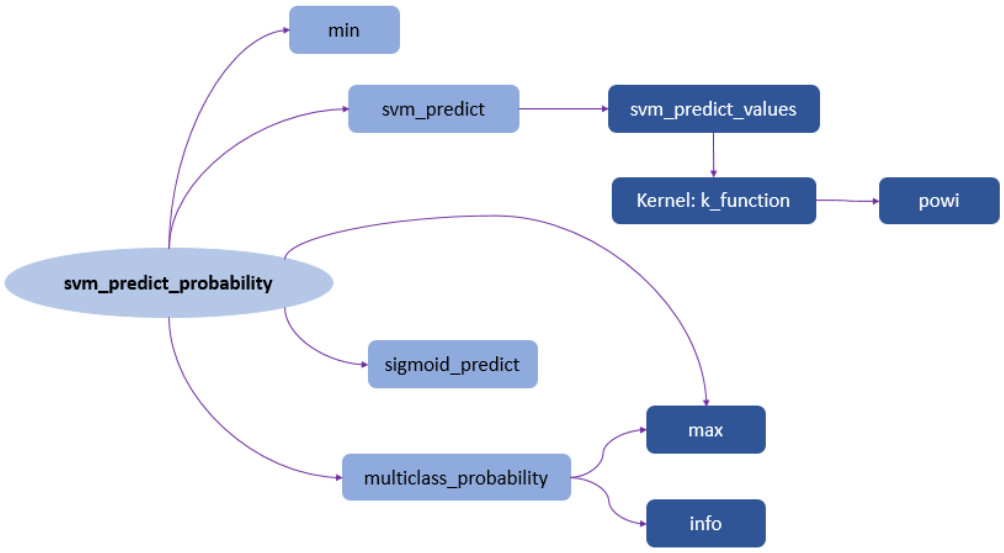

Figura 23. Esquema de dependencia de funciones de svm_predict_probability()

En la Figura 24 se comprueba cómo *svm_predict()* realiza la llamada a la función *svm_predict_values(),* que es la encargada de proporcionar valores de decisión sobre un vector de prueba a partir del modelo, devolviendo la clase predicha.

```
double svm_predict(const svm_model *model, const svm_node *x)
{
  int nr_class = model->nr_class;
  double *dec_values;
  if(model->param.svm_type == ONE_CLASS ||
     model->param.svm_type == EPSILON_SVR ||
     model->param.svm_type == NU_SVR)
    dec_values = Malloc(double, 1);
  else
    dec_values = Malloc(double, nr_class*(nr_class-1)/2);
  double pred_result = svm_predict_values(model, x, dec_values);
  free(dec_values);
  return pred_result;
}
```

Figura 24. Código de la función svm_predict()

Para ello, la función *svm_predict_values()*, a partir de un modelo de clasificación con *nr_class* clases, proporciona *nr_class*(nr_class-1)/2* valores de decisión en la matriz *dec_values*, tal y como se muestra en el código de la Figura 25.

```
double svm_predict_values(const svm_model *model, const svm_node *x, double* dec_values)
{
  int i;
  if(model->param.svm_type == ONE_CLASS ||
     model->param.svm_type == EPSILON_SVR ||
     model->param.svm_type == NU_SVR)
  {
    double *sv_coef = model->sv_coef[0];
    double sum = 0;
    for(i=0;i<model->l;i++)
      sum += sv_coef[i] * Kernel::k_function(x,model->SV[i],model->param);
    sum -= model->rho[0];
    *dec_values = sum;

    if(model->param.svm_type == ONE_CLASS)
      return (sum>0)?1:-1;
    else
      return sum;
  }
  else
  {
    int nr_class = model->nr_class;
    int l = model->l;

    double *kvalue = Malloc(double,l);
    for(i=0;i<l;i++)
      kvalue[i] = Kernel::k_function(x,model->SV[i],model->param);
```

```
int *vote = Malloc(int,nr_class);
for(i=0;i<nr_class;i++)
  vote[i] = 0;

int p=0;
for(i=0;i<nr_class;i++)
  for(int j=i+1;j<nr_class;j++)
  {
    double sum = 0;
    int si = start[i];
    int sj = start[j];
    int ci = model->nSV[i];
    int cj = model->nSV[j];

    int k;
    double *coef1 = model->sv_coef[j-1];
    double *coef2 = model->sv_coef[i];
    for(k=0;k<ci;k++)
      sum += coef1[si+k] * kvalue[si+k];
    for(k=0;k<cj;k++)
      sum += coef2[sj+k] * kvalue[sj+k];
    sum -= model->rho[p];
    dec_values[p] = sum;

    if(dec_values[p] > 0)
      ++vote[i];
    else
      ++vote[j];
    p++;

      }

    int vote_max_idx = 0;
    for(i=1;i<nr_class;i++)
      if(vote[i] > vote[vote_max_idx])
        vote_max_idx = i;

    free(kvalue);
    free(start);
    free(vote);
    return model->label[vote_max_idx];
  }
}
```

Figura 25. Código de la función svm_predict_values()

La función *svm_predict_values()* hace uso de la función *kernel*, que recoge los distintos casos posibles en función del *kernel* a utilizar. El fragmento del código de esta función, referente al tipo de *kernel* RBF, se muestra en la Figura 27. La función del núcleo RBF calcula, para dos puntos dados, la similitud o la cercanía que existe entre ellos. El núcleo se puede representar siguiendo la ecuación que se muestra en la Figura 26, donde σ es la varianza del *kernel* y $||x_1 - x_2||$ representa la distancia euclídea entre los dos puntos. En el caso del *kernel* RBF el espacio sobre el que se calcula el producto escalar entre vectores es un espacio de dimensiones infinitas.

$$K(x_1, x_2) = \exp\left(-\frac{||x_1 - x_2||^2}{2\sigma^2}\right)$$

Figura 26. Función del núcleo RBF

En función de la distancia euclídea a la que se encuentre una observación con respecto a otra, el *kernel* asignará un valor mayor o menor. Por lo que el *kernel* RBF representa la similitud entre dos vectores como una función que decrece según la distancia euclídea entre estos. El valor máximo que puede tener este *kernel* es 1, cuando los dos puntos son iguales, no hay distancia entre ellos y son extremadamente similares. Sin embargo, si los puntos están separados una gran distancia, entonces el valor del *kernel* es menor que 1 y cercano a 0.

```
case RBF:
{
  double sum = 0;
  while(x->index != -1 && y->index !=-1)
  {
    if(x->index == y->index)
    {
      double d = x->value - y->value;
      sum += d*d;
      ++x;
      ++y;
    }
    else
    {
      if(x->index > y->index)
      {
        sum += y->value * y->value;
        ++y;
      }
      else
      {
        sum += x->value * x->value;
        ++x;
      }
    }
  }

  while(x->index != -1)
  {
    sum += x->value * x->value;
    ++x;
  }

  while(y->index != -1)
  {
    sum += y->value * y->value;
    ++y;
  }

  return exp(-param.gamma*sum);
}
```

Figura 27. Código de la función Kernel::k_function()

CAPÍTULO 4 Parámetros y señales electromiográficas

En este capítulo se presenta una introducción teórica sobre las señales electromiográficas, al representar buena parte de los datos que se van a recoger en la plataforma *hardware/*software que se va a desarrollar y, en consecuencia, por su gran relevancia dentro del presente Trabajo Fin de Grado. A partir del estudio sobre las señales EMG, se plantea el análisis realizado sobre una serie de parámetros y características que permitan procesar los valores capturados de una manera más eficiente, eliminando la necesidad de manejar los datos en crudo de forma directa. Se justifican las decisiones tomadas en cuanto a los parámetros seleccionados y, una vez presentadas las características elegidas, se presentan diversas pruebas iniciales realizadas con el fin de obtener una primera aproximación de resultados y verificar que realmente son parámetros válidos para la detección de los gestos.

4.1 Señales electromiográficas

Las señales electromiográficas (EMG) son señales eléctricas producidas por un músculo durante el proceso de contracción y relajación. Es decir, están directamente relacionadas con la respuesta a un movimiento muscular, donde el nivel de esfuerzo queda determinado por el número de fibras musculares que se activan por una neurona durante la contracción. El proceso de medición de estas señales eléctricas se conoce como Electromiografía. La señal EMG proporciona información sobre el funcionamiento total del sistema motor, es decir, sobre las uniones neuromusculares. Esta sinapsis neuromuscular es la unión entre el axón de una neurona y las fibras musculares. Por tanto, los músculos del brazo reciben las órdenes para su movimiento desde el cerebro a través de impulsos eléctricos. Las señales EMG constituyen una representación específica de la intención del movimiento que ejecuta un usuario al realizar un gesto de la mano [30].

Una neurona motora o motoneurona es la neurona que emite el impulso que provoca la contracción de la fibra muscular. Es, por tanto, la encargada de conducir los impulsos del cerebro y la médula espinal hacia los músculos. La neurona motora, su axón y el conjunto de fibras musculares que se estimulan constituyen la unidad motora, como se muestra en la Figura 28. Estas unidades son responsables de la fuerza y el movimiento de los músculos.

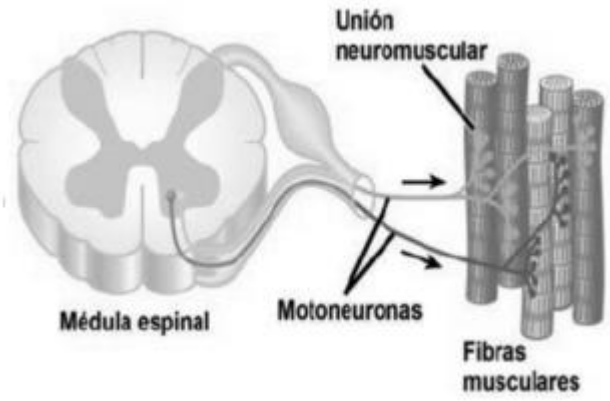

Figura 28. Unidad motora [31]

Un músculo está formado por varias unidades motoras y estas, a su vez, están formadas por fibras musculares. Cada unidad motora es inervada por una única motoneurona. En la Figura 29 se pueden observar dos unidades motoras, identificadas en color violeta y rojo, respectivamente.

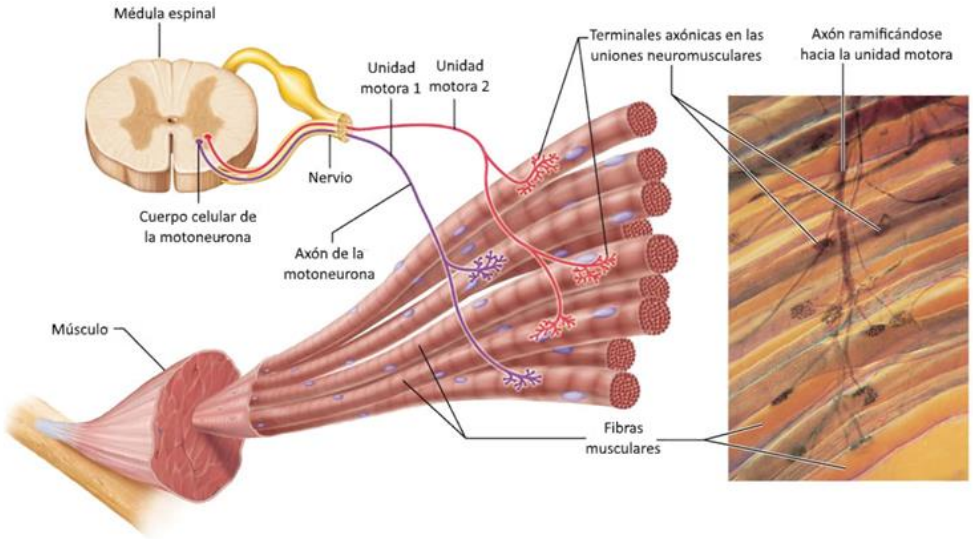

Figura 29. Estructura muscular [32]

Cuando una neurona motora se activa, se propagan a lo largo de cada fibra ondas conocidas como trenes de potencial de acción. Los potenciales de acción (MUAP, *Motor Unit Action Potencial*) de las unidades motoras son pulsos eléctricos que contienen parámetros químicos característicos y que se transfieren por las fibras musculares a intervalos irregulares [33]. El conjunto de estas señales o pulsos eléctricos constituyen el tren de acciones de potencial o MUAPT (*Motor Unit Action Potencial Train*). En la Figura 30 se representa de forma gráfica el proceso de la unión neuromuscular y los elementos que intervienen en ella.

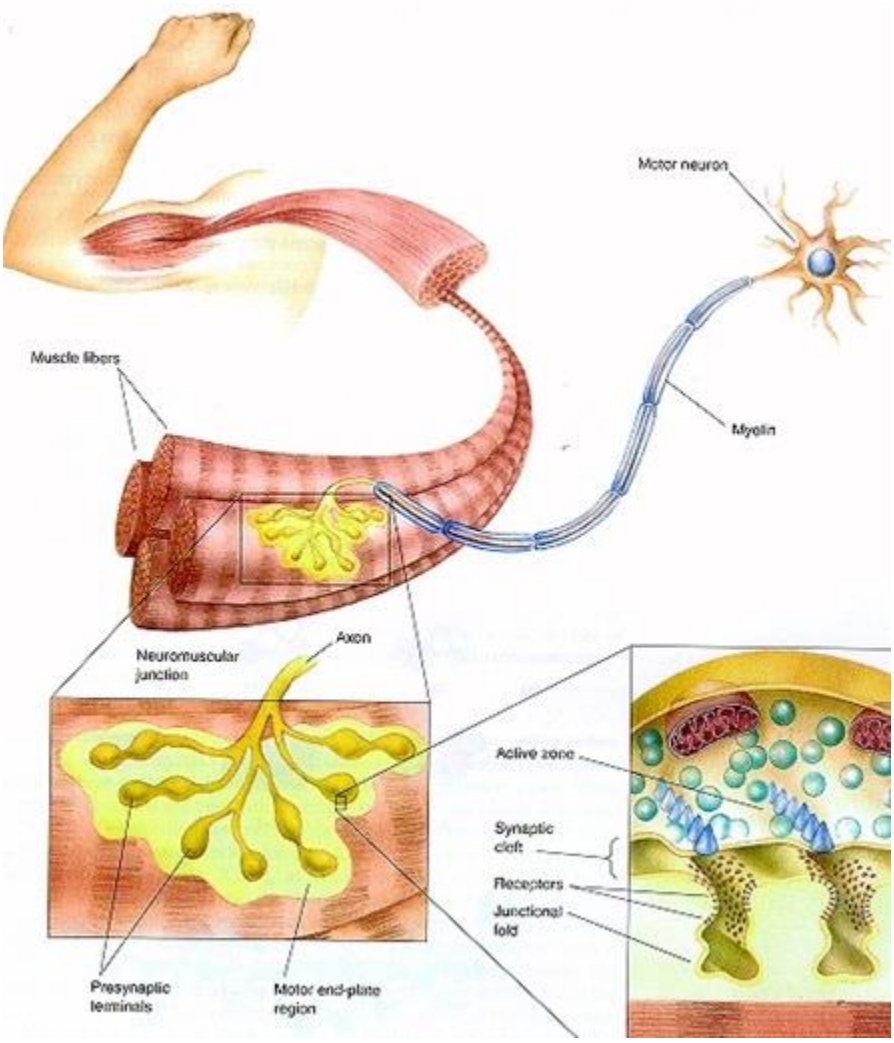

Figura 30. Proceso de la unión neuromuscular [34]

A continuación, puede describirse cómo se produce el proceso de contracción muscular. Las neuronas o células nerviosas que liberan los estímulos que dan lugar a las contracciones constan de un terminal llamado axón, que se ramifica en el músculo en terminales axonales. Cuando el impulso nervioso se transfiere a través de la neurona motora y llega a la unión entre esta y el músculo, la neurona libera un compuesto llamado acetilcolina. Este neurotransmisor posibilita el paso de un impulso nervioso desde los terminales del axón al músculo. El compuesto se difunde a través de la unión entre la neurona y la fibra muscular, y se combina con receptores de la fibra. Como respuesta, se produce un cambio eléctrico en la membrana celular, denominado despolarización, que inicia el impulso eléctrico o potencial de acción [35].

Durante la contracción muscular se producen dos estados principales en las señales EMG, un estado estacionario y otro estado transitorio. En el estado estacionario la contracción muscular es sostenida y no se modifica. Esto se produce cuando las unidades motoras involucradas en el movimiento del gesto y la contracción muscular ya se encuentran activas de

forma estable. En cambio, en el modo transitorio se produce una contracción muscular repentina y no sostenida, que se corresponde con el inicio de la actividad de las unidades motoras involucradas en el movimiento. Pueden clasificarse como señales EMG en estado estacionario los gestos de larga duración, al ser sostenidos y no modificarse en el transcurso del tiempo. Sin embargo, los gestos de corta duración se consideran señales EMG en estado transitorio. Los gestos de larga duración, al presentar una naturaleza menos compleja, alcanzan un mayor porcentaje de exactitud en el reconocimiento y clasificación de las señales EMG.

Con estas señales se puede conseguir crear interfaces de comunicación entre usuario y máquina. La captura de las señales electromiográficas de los músculos se produce mediante el uso de electrodos. Para obtener los datos de las señales musculares, estos electrodos detectan los campos eléctricos variables en el tiempo que producen los cambios electroquímicos de la fibra muscular. Muchos estudios evidencian que la señal EMG tiene una amplitud típica entre 0 y 10 mV, y su frecuencia útil está en el rango de 0 a 500 Hz con la mayor cantidad de energía concretada entre los 50 y 150 Hz. En la Figura 31 se presenta la gráfica que refleja la energía del espectro de una señal electromiográfica aleatoria, donde se observa cómo los mayores picos de energía se encuentran por debajo de los 200 Hz. Dentro de este rango dominante, concretamente a 60 Hz, el cuerpo humano es una potente antena para el ruido ambiental [36]. Por eso, en muchas ocasiones es necesario utilizar etapas de amplificación y filtrado que eliminen el ruido y otras señales que no son de interés para el análisis de las señales electromiográficas.

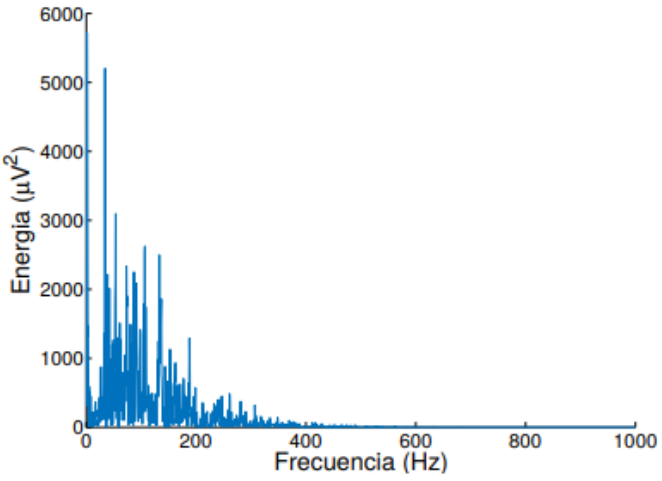

Figura 31. Espectro de frecuencias de una señal de electromiografía [37]

Algunos ejemplos de las diferentes salidas que se pueden obtener a partir de los sensores se muestran en la Figura 32. En la gráfica superior la señal EMG sin procesar, en la que se presentan los potenciales de acción de todas las unidades motoras detectables en la superficie

del electrodo donde se superponen eléctricamente, mostrando una distribución de amplitudes positivas y negativas. En la siguiente gráfica se identifica la señal EMG rectificada, que tras aplicar un algoritmo donde los valores negativos se convierten a positivos, ve reflejados hacia hacia arriba sus picos negativos. La gráfica inferior muestra una señal EMG rectificada e integrada, a la que se le han aplicado filtros digitales para obtener la envolvente de la señal.

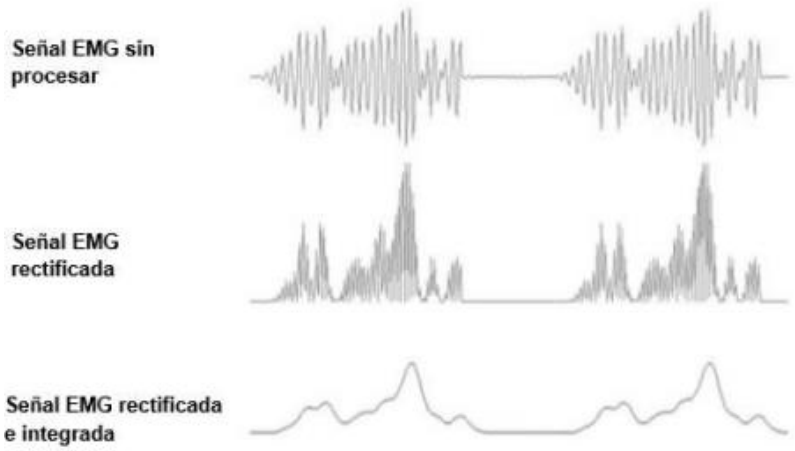

Figura 32. Ejemplo de señal EMG [31]

Existen dos métodos principales para la detección de las señales EMG, uno superficial y otro intramuscular [35]. Para el EMG intramuscular se utiliza una aguja hipodérmica que se introduce en la piel hasta el tejido muscular con el fin de poder detectar la actividad eléctrica. Cada trazo del electrodo da una imagen muy local de la actividad de un músculo concreto. Por eso, la aguja debe introducirse en diferentes localizaciones para obtener resultados confiables, ya que el músculo esquelético difiere en su estructura interna. La electromiografía intramuscular o de aguja se realiza con el fin de estudiar la patología o fisiología de las unidades motrices. Como alternativa a este método invasivo, se plantea el método superficial, en el que, sin necesidad de atravesar la piel, se utiliza una superficie en la que el electrodo pueda controlar la imagen general de la activación muscular, a diferencia de la activación de solo unas pequeñas fibras. La electromiografía de superficie se enfoca a estudios relacionados con el comportamiento muscular, patrones de actividad temporal o fatiga muscular. Concretamente, para el desarrollo de este TFG se tendrá en cuenta la electromiografía de superficie, que permite una gran facilidad de colocación de los electrodos y, además, presenta un carácter no invasivo al colocarse en contacto sobre la piel.

El estudio de las señales electromiográficas ha permitido el desarrollo de numerosos proyectos relevantes en el ámbito del bienestar humano. Destacan, entre ellos, las líneas de investigación para el entrenamiento de prótesis y predicción de fatiga en los músculos, así como

la optimización en el diseño de los algoritmos de reconocimiento de patrones en tiempo real. La detección, el análisis y el procesamiento de las señales EMG siguen siendo ampliamente utilizados en diversos campos como el diagnóstico clínico, rehabilitación clínica, investigación básica y aplicada o desarrollo tecnológico. En el presente TFG se ha utilizado un dispositivo comercial, que ya incluye un circuito de adquisición y amplificación de señales EMG. Esto supone una gran ventaja, ya que la manipulación y el acondicionamiento de las señales de los electrodos son algunos de los aspectos que mayores inconvenientes pueden reportar.

4.2 Estudio previo y análisis de características para el procesamiento de las señales EMG

Para la adquisición de las señales electromiográficas en el presente TFG, se utiliza el brazalete *Myo Gesture Control Armband*. Este dispositivo está formado por ocho sensores EMG que funcionan como electrodos de superficie, representan el potencial eléctrico de los músculos como respuesta a la activación muscular, gracias a las señales EMG proporcionadas por los sensores. La distribución de estos ocho sensores en el dispositivo se puede visualizar en la Figura 33. Sin embargo, debido al bajo potencial eléctrico del músculo, las señales son sensibles a otras fuentes de ruido. La gama de potenciales proporcionados por el brazalete *Myo* se encuentra entre -128 y 128 en unidades de activación, que son valores enteros de la amplificación de los potenciales medidos por los sensores EMG [38]. La velocidad de muestreo a la que el dispositivo *Myo* es capaz de extraer los datos EMG es de 200 Hz.

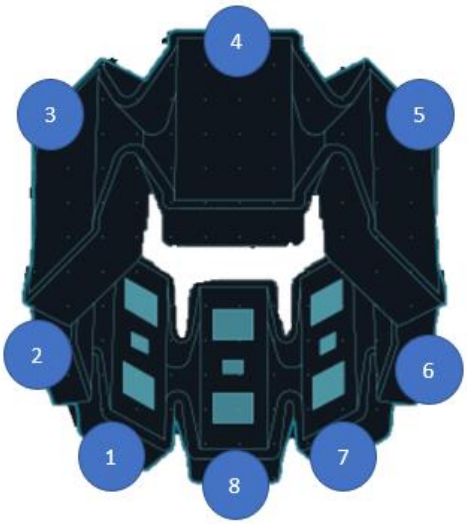

Figura 33. Disposición de sensores en el brazalete Myo

Además de los ocho sensores EMG, el brazalete dispone también de una unidad de medición inercial (IMU) de nueve ejes, constituido por un acelerómetro, un magnetómetro y un

giroscopio, cada uno de ellos de tres ejes. Gracias a esta IMU se puede determinar el movimiento o la orientación de la mano, a partir de los datos de *pitch*, *roll* y *yaw*. La extracción de datos IMU se produce con una frecuencia de muestreo de 50 Hz. En la Figura 34 se representan algunos de los elementos que componen el brazalete *Myo*, entre los que se destacan los sensores EMG.

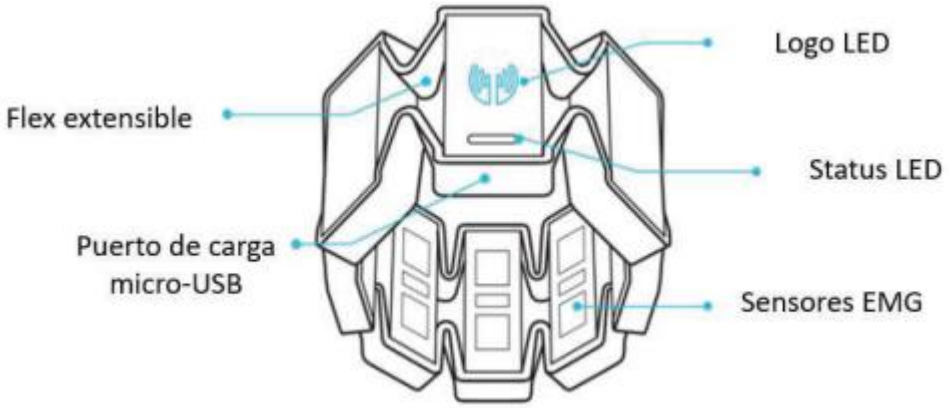

Figura 34. Representación de elementos en el dispositivo Myo [16]

Todo esto constituye el conjunto de datos en crudo que se pueden obtener a partir del dispositivo utilizado, y desde los que se parte inicialmente en este TFG. Sin embargo, considerar todos estos valores sin procesar en cada una de las muestras para realizar el reconocimiento de los gestos podría resultar muy complejo e ineficiente. Por ello, es indispensable realizar un estudio previo que contemple qué parámetros pueden resultar los idóneos o convenientes para la detección e identificación de gestos. Para ello, se debe determinar qué cálculos aportan resultados más positivos para el proceso de clasificación a partir de los datos de los sensores EMG e IMU. Esto es, seleccionar aquellos con los que, a partir de un menor número de parámetros, se obtengan buenos resultados. Un menor número de parámetros a tener en cuenta supone la generación de un modelo de menor tamaño y, por tanto, una reducción de problemas de memoria en el dispositivo, ente otros factores. Esto determina un aspecto muy relevante, ya que tendrá un impacto directo en la complejidad de la plataforma *hardware/software*, el tiempo de procesamiento, la memoria del dispositivo de IoT basado en MCU o el número de gestos que van a poder reconocerse. Si la mayor parte de la memoria del dispositivo se requiere para la programación y el almacenamiento las tablas de clasificación (que dependen del número de parámetros que se utilizan para las predicciones) se limita el número de gestos que pueden incluirse para reconocer.

En la Figura 35 se representa un diagrama del proceso que se lleva a cabo para el cálculo de los parámetros. A la izquierda se representa el brazalete *Myo Armband*, del que se extraen los datos en crudo proporcionados por los sensores y la unidad de medición inercial que integra.

Las señales del dispositivo se envían inalámbricamente mediante una comunicación BLE con el dispositivo de IoT. A partir de ahí, se realiza el procesamiento correspondiente de los valores extraídos y se realizan los cálculos de los parámetros que se utilizarán para la generación de los modelos.

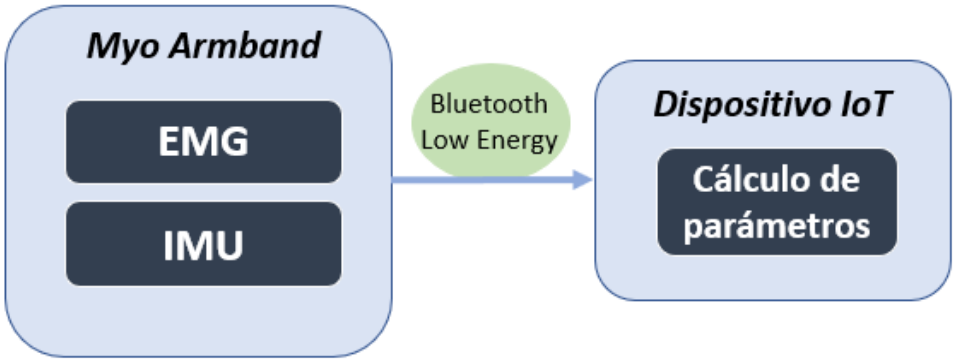

Figura 35. Diagrama para el cálculo de parámetros

El tipo de información que se obtiene de las señales EMG se puede definir como señales aleatorias, ya que presentar incertidumbre en sus parámetros y no pueden ser descritas como funciones matemáticas concretas. Por ese motivo, requieren de un análisis mediante el uso de técnicas estadísticas que permitan un procesamiento adecuado en el dominio del tiempo. Para poder decidir qué cálculos aplicar a los datos en su procesamiento, se ha llevado a cabo una búsqueda y estudio en profundidad de los diferentes parámetros y técnicas que se han empleado en proyectos similares o artículos de investigación que también tienen en cuenta sensores EMG e IMU [37] [39] [3]. A partir de la información recabada, se presentan algunas de las técnicas más utilizadas para llevar a cabo el procesamiento de las señales EMG.

Uno de los parámetros más importantes es el del valor medio absoluto, que se utiliza para calcular el promedio del valor absoluto de la señal EMG. Está relacionado con los puntos de contracción del músculo en cada uno de los instantes de tiempo. La ecuación que determina su cálculo se presenta en la Figura 36, donde N denota el tiempo de muestreo de los datos en bruto, n es el n-ésimo canal del dispositivo *Myo,* y $E_n(t)$ representa un dato entero de 8 bits que corresponde a una señal EMG del n-ésimo canal.

$$EMG_{avg}(n) = \frac{1}{N}\sum_{t=1}^{N}|E_n(t)|$$

Figura 36. Fórmula EMG_{avg} [39]

A raíz de este cálculo se plantean también parámetros como la suma de los promedios de EMG de cada canal del brazalete *Myo*. Se denomina EMG_{sum} y su fórmula se muestra en la

Figura 37. Una alternativa es la del cálculo de la media de todos los valores de EMG_{avg}. En la Figura 38 se muestra la fórmula de este parámetro, denominado EMG_{avg_sum}.

$$EMG_{sum} = \sum_{n=1}^{8} EMG_{avg}(n)$$

Figura 37. Fórmula EMG_{sum} [39]

$$EMG_{avg\_sum} = \frac{1}{N} \sum_{n=1}^{N} EMG_{avg}(n)$$

Figura 38. Fórmula EMG_{avg_sum}

Otro de los posibles parámetros es la desviación estándar, que indica la variación los datos con respecto a la media o a un valor central [40]. La fórmula asociada a su cálculo se presenta en la Figura 39, donde N representa la longitud de la señal EMG, x_i representa la señal EMG en un segmento i, y $\bar{x}_i$ representa la media de la señal EMG en un segmento i.

$$STD = \sqrt{\frac{1}{N} \sum_{i=1}^{N-1} (x_i - \bar{x})^2}$$

Figura 39. Fórmula de la desviación estándar [40]

En otras ocasiones se plantea también la utilización del parámetro de la energía, que indica la potencia total que tiene la señal EMG, y representa un indicador de la fuerza con la que se contraen o relajan los músculos en función del gesto realizado [40]. En la Figura 40 se indica la fórmula para el cálculo de la energía de la señal. De nuevo, N representa la longitud de la señal EMG y x_i representa la señal EMG en un segmento i.

$$MVA = \frac{1}{N} \sum_{i=1}^{N} |x_i|$$

Figura 40. Fórmula de la energía de la señal [40]

También, puede tenerse en cuenta el cálculo de la raíz cuadrática media (RMS), que es una medida de la posición central del conjunto de valores. Caracteriza la señal de acuerdo con la estimación del contenido de energía, en relación con su amplitud, en un determinado intervalo de tiempo [41]. Este parámetro está relacionado con la indicación de contracción

muscular con fuerza constante antes de comenzar la fatiga muscular. La media cuadrática es muy útil para calcular la media de variables que toman valores negativos y positivos. Se suele utilizar cuando el símbolo de la variable no es relevante e interesa el valor absoluto del elemento. La fórmula necesaria para el cálculo se muestra en la Figura 41, donde N y x_i representan nuevamente los mismos factores descritos hasta ahora.

$$RMS = \sqrt{\frac{1}{N}\sum_{i=1}^{N} {x_i}^2}$$

Figura 41. Fórmula de la raíz cuadrática media [40]

La varianza es el promedio del valor cuadrado de la amplitud de la señal EMG. Su ecuación se define en la Figura 42, donde L es la longitud del segmento del tiempo, x_j es el valor de cada parte de la muestra y $\bar{x}$ es la media de la señal.

$$VAR = \frac{1}{L}\sum_{j=1}^{L}\left(x_j - \bar{x}\right)^2$$

Figura 42. Fórmula de la varianza [41]

Por otro lado, el análisis de la señal de EMG en el dominio de la frecuencia contiene medidas y parámetros que describen perfiles del espectro de frecuencias de la señal electromiográfica. Técnicas como la FFT (*Fast Fourier Transform*) son ampliamente utilizadas para obtener el espectro de la señal de EMG [37]. Sin embargo, se determina que únicamente se va a trabajar en el dominio temporal, no aplicando en ningún caso aspectos del dominio frecuencial. Asumiendo que no sea siempre la opción óptima, debido a la complejidad de cómputo se opta por trabajar en el dominio del tiempo. Se pretende conseguir una solución de bajo coste, por lo que no se busca la máxima optimización. Es por ello por lo que se han descartado directamente aquellos parámetros basados en la frecuencia, como la energía de la señal, densidad del espectro de potencia, frecuencia media, frecuencia mediana, transformada discreta de Fourier, etc.

Adicionalmente, se intentaron establecer otros métodos de clasificación válidos para no tener en cuenta únicamente los parámetros individuales de los sensores. Se buscaban otros posibles cálculos que combinasen los datos de los ocho EMG en un único parámetro. De esta forma se sustituirían los ocho valores de las ocho medias del valor absoluto de los EMG para procesar, por ocho parámetros como, por ejemplo, la suma de las ocho medias, la energía de las ocho medias, el valor absoluto cuadrático medio de las ocho medidas de las EMG, etc. Para

un mejor entendimiento, el planteamiento descrito anteriormente se refleja de manera gráfica en la Figura 43, en la que se muestra el esquema de dos posibles vectores que se utilizarían para la clasificación de los gestos en función de los diferentes parámetros seleccionados. En el esquema superior, cada uno de los elementos del vector está formado por el valor medio tomando la media del valor absoluto de la señal EMG de uno de los canales en cada instante de tiempo, mientras que, en el esquema inferior, cada uno de los elementos del vector está formado por parámetros calculados a partir de los valores de las medias que conforman el primer vector.

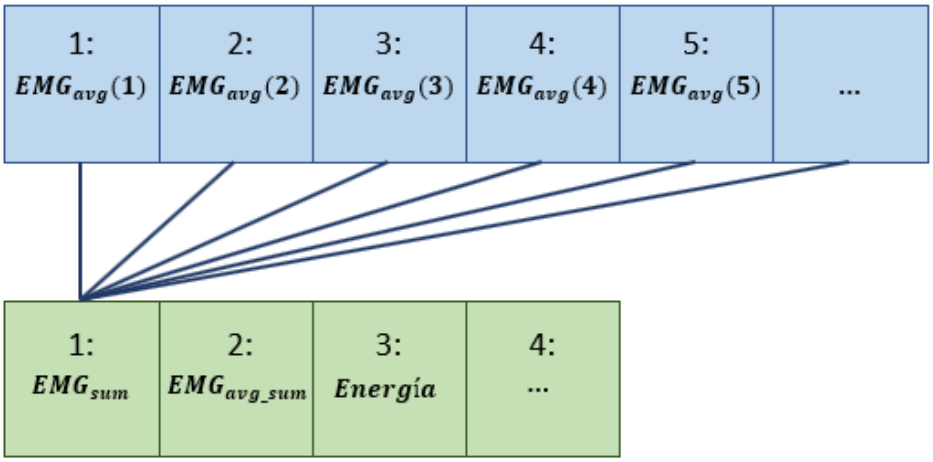

Figura 43. Ejemplo de vectores para la clasificación

De esta manera se planteaba un supuesto más eficiente, seleccionando parámetros que individualmente pudiesen tener una mayor relevancia y que minimizasen el número de entradas para el proceso de clasificación. Sin embargo, se decidió no descartar la información que directamente aporta cada uno de los parámetros de EMG_{avg}, y se determinó que las medias de los valores de cada uno de los canales serían siempre parámetros a contemplar.

Por tanto, inicialmente los parámetros que se considera tener en cuenta en el proceso de clasificación serán las ocho medias de los valores absolutos de los EMG en cada instante de tiempo (EMG_{avg}), la media de los ocho valores de EMG_{avg} (EMG_{avg_sum}) o, en su defecto, la suma de los ocho valores de EMG_{avg} (EMG_{sum}), el *pitch* y el *roll* del acelerómetro, y las desviaciones estándar de x, y, z.

4.3 Análisis inicial con datos experimentales

Para una primera aproximación se ha partido de un conjunto de datos experimentales de prueba que permitan obtener un primer análisis de resultados. De esta manera, se consigue no abarcar directamente la toma de muestras con el dispositivo *Myo Armband* y realizar

diferentes pruebas con los datos y valores calculados para, en una primera instancia, comprobar si los parámetros elegidos pueden resultar adecuados para la clasificación. Estas muestran son aportadas por los colaboradores Leobardo Sánchez Velasco, Manuel Arias-Montiel y Enrique Guzmán Ramírez, de la Universidad Tecnológica de la Mixteca. Esta información proporciona datos EMG obtenidos a partir de un dispositivo *Myo Armband* en respuesta a ocho gestos de la mano utilizados para controlar una prótesis. Los datos están divididos en ocho archivos. Cada uno de ellos consta de cincuenta lecturas del brazalete *Myo* para un gesto definido, y cada lectura incluye cien muestras por sensor [42]. La señal EMG se muestrea a una frecuencia de 50 muestras por segundo. En la Figura 44 se puede ver, a modo de ejemplo, una de las cincuenta lecturas correspondientes al Gesto 1. Para una mejor legibilidad solo se muestran las 25 primeras muestras de cada uno de los ochos sensores, en lugar de las cien que se presentan por cada línea. A continuación, se especifica qué acción se corresponde a cada uno de los identificadores de los gestos.

***HandGesture01*:** gesto de la mano que genera el agarre cilíndrico en la prótesis.

***HandGesture02*:** gesto de la mano que genera el agarre de la punta en la prótesis.

***HandGesture03*:** gesto de la mano que genera el agarre del gancho en la prótesis.

***HandGesture04*:** gesto de la mano que genera la mano abierta en la prótesis.

***HandGesture05*:** gesto de la mano que genera el agarre palmar en la prótesis.

***HandGesture06*:** gesto de la mano que genera el agarre esférico en la prótesis.

***HandGesture07*:** gesto de la mano que genera el agarre lateral en la prótesis.

***HandGesture08*:** gesto de la mano que genera el puño en la prótesis.

```
// 1 Reading of the 8 sensors, 100 samples per sensor
{{194, 182, 187, 202, 200, 221, 240, 229, 230, 236, 213, 198, 226, 259, 254, 256, 285, 274, 232, 217, 207, 187, 211, 205, 210,
{419, 437, 383, 407, 440, 426, 432, 433, 427, 424, 407, 458, 468, 485, 455, 436, 381, 365, 317, 319, 295, 247, 322, 331, 365,
{170, 180, 174, 202, 201, 210, 191, 197, 222, 240, 229, 250, 260, 217, 203, 222, 207, 181, 166, 158, 119, 102, 111, 121, 147,
{552, 492, 477, 473, 382, 510, 523, 539, 506, 526, 413, 449, 452, 533, 548, 610, 635, 644, 668, 664, 598, 667, 728, 661, 696,
{360, 343, 339, 354, 325, 372, 378, 378, 337, 340, 289, 298, 292, 342, 356, 372, 368, 392, 481, 479, 454, 498, 502, 382, 404,
{110, 106, 117, 121, 119, 131, 130, 120, 118, 134, 113, 126, 123, 122, 112, 121, 113, 130, 180, 182, 182, 189, 189, 137, 129,
{267, 284, 280, 281, 297, 258, 226, 239, 266, 293, 291, 314, 303, 285, 272, 307, 297, 305, 317, 296, 328, 344, 352, 355, 384,
{271, 271, 265, 272, 294, 295, 289, 301, 377, 379, 372, 378, 404, 321, 357, 356, 394, 362, 378, 336, 327, 285, 286, 288, 314,
```

Figura 44. Ejemplo de una de las lecturas del Gesto1 de los datos experimentales

Además, en la Figura 44 se puede comprobar cómo los valores aportados en cada uno de los ficheros han sido previamente escalados y rectificados, al representar valores de hasta 124 y contar solo con valores positivos y ningún valor menor que cero.

A partir de este punto, con el fin de realizar el tratamiento de los datos, su procesamiento y los cálculos necesarios para obtener los parámetros correspondientes, se plantearon dos estrategias diferentes. Una de ellas consistía en el procesamiento con la herramienta de *Microsoft Excel* y otra mediante *scripts* de Python. En este caso, para llevar a

cabo las comprobaciones a modo de primera aproximación solo se tendrán en cuenta los parámetros asociados a los sensores EMG y no otros como el *pitch*, *roll* o desviaciones estándar.

4.3.1 Cálculos realizados en Excel

En primer lugar, se procesaron los datos experimentales para conseguir los parámetros buscados utilizando hojas de cálculo Excel. Se tomaron cuatro de los ocho gestos disponibles en los datos experimentales: *HandGesture01*, *HandGesture02*, *HandGesture03* y *HandGesture04*. Asimismo, de las cincuenta lecturas con las que se cuenta de cada gesto, se tomaron veinticinco aleatorias para el entrenamiento, y ocho aleatorias para la validación.

Para llevar a cabo los cálculos se volcaron en tablas las cien muestras de cada uno de los sensores EMG para cada una de las veintiocho lecturas por cada gesto. A partir de esto, se calcularon las medias de las cien muestras en valor absoluto de cada canal en cada caso. Este cálculo, anteriormente mencionado en la Figura 36, se ha denominado *EMG Average* (EMG_{avg}). A partir de los valores obtenidos de EMG_{avg} se obtuvo el parámetro denominado EMG_{sum}, cuya fórmula puede recordarse en la Figura 37, y que se corresponde con la suma de las medias calculadas para cada uno de los EMG.

En la Figura 45 se representa un extracto de una de las tablas correspondiente a una de las lecturas aleatorias de *HandGesture01*. Para una mejor legibilidad se muestran únicamente veinticuatro muestras en cada una de las filas, en lugar de las cien que se utilizan realmente para los cálculos.

Reading nº1	100 samples																								EMG Average	EMG sum
CH1	194	182	187	202	200	221	240	229	230	236	213	243	...	248	246	240	211	212	203	210	231	244	257	289	264	228,28	
CH2	419	437	383	407	440	426	432	433	427	424	407	412	...	383	349	333	350	342	323	346	365	347	369	392	366	364,13	
CH3	170	180	174	202	201	210	191	197	222	240	229	232	...	219	213	217	220	218	212	208	193	193	181	196	208	198,59	
CH4	552	492	477	473	382	510	523	539	506	526	413	729	...	645	540	523	508	566	561	565	603	558	530	594	631	609,15	2716,12
CH5	360	343	339	354	325	372	378	378	337	340	289	563	...	514	448	412	417	443	463	441	449	447	396	434	450	436,02	
CH6	110	106	117	121	119	131	130	120	118	134	113	197	...	193	178	174	162	161	163	155	165	156	143	163	170	152,11	
CH7	267	284	280	281	297	258	226	239	266	293	291	374	...	473	548	536	563	560	500	384	403	363	362	401	446	338,36	
CH8	271	271	265	272	294	295	289	301	377	379	372	450	...	503	528	543	510	470	465	403	422	404	442	453	455	389,48	

Figura 45. Tabla para el cálculo de parámetros

A continuación, los resultados de los cálculos para cada uno de los cuatro gestos se volcaron en una tabla final para obtener en cada una de las filas los nueve datos de interés. En la Figura 46 se puede consultar la tabla que recoge los datos finales correspondientes a *HandGesture01*. En color verde se diferencian las lecturas destinadas para el entrenamiento y en color naranja las lecturas que se utilizan para la validación. Por tanto, cada fila corresponde a cada una de las veintiocho lecturas aleatorias escogidas. En las ocho primeras columnas se representa la media calculada de las cien muestras tomadas por cada uno de los ocho sensores

EMG. Finalmente, la última columna expresa el valor de la suma de las medias de los EMG calculadas.

HAND GESTURE 1									
Reading	EMG avg								EMG sum
	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8	
1	228,28	364,13	198,59	609,15	436,02	152,11	338,36	389,48	2716,12
2	257,58	304,57	212,42	587,95	409,65	148,84	393,35	505,89	2820,25
3	296,66	402,68	160,23	711,31	525,83	191,43	429,97	594,39	3312,5
4	358,22	547,87	195,11	664,57	500,58	171,31	398,85	511,74	3348,25
5	230,42	318,89	149,22	548,08	414,93	142,87	305,55	347,81	2457,77
6	325,15	506,04	283,98	617,55	475,11	185,59	459,3	581,42	3434,14
7	229,89	343,62	137,48	545,54	403,32	144,17	328,42	331,72	2464,16
8	287,23	442,65	179,68	641,43	465,12	167,68	362,7	430,45	2976,94
9	319,95	462,18	233,74	637,53	462,69	179,79	465,15	551,26	3312,29
10	339,38	512,56	200,26	608,46	462,07	177,77	394,43	502,36	3197,29
11	266,79	392,85	202,37	553,66	389,67	142,05	365,18	464,05	2776,62
12	185,73	263,01	120,97	452,78	303,86	113,65	283,84	321,45	2045,29
13	228,73	345,2	159,08	524,75	369,9	142,36	334,9	364,02	2468,94
14	211,92	302,84	194,41	486,03	337,12	121,01	316,67	361,31	2331,31
15	211,84	279,43	156,04	558,68	398,95	143,02	310,24	339,07	2397,27
16	224,17	293,98	174,69	524,53	368,87	132,87	349,93	411,37	2480,41
17	188,92	264,51	121,24	534,51	380,25	135,35	298,19	309,11	2232,08
18	199,96	306,93	135,21	493,13	373,64	135,82	292,88	340,47	2278,04
19	188,07	236,61	154,12	512,11	361,69	125,55	260,55	309,23	2147,93
20	196,01	275,17	144,02	512,76	368,18	125,21	303,81	339,44	2264,6
21	234,27	325,94	171,12	576,27	402,98	149,14	341,96	371,11	2572,79
22	242,73	345,38	165,11	516	357,84	136,09	321,87	375,74	2460,76
23	196,66	272,74	186,11	460,64	336,49	121,7	297,86	287,84	2160,04
24	230,68	297,68	169,64	560,27	401,44	140,86	361,15	401,91	2563,63
25	255,27	346,63	168,81	646,56	456,05	157,47	374,13	436,79	2841,71
26	219,76	315,5	156,76	587,36	404,05	141,9	336,79	352,88	2515
27	220,13	308,18	190,24	519,6	358,24	125,92	310,93	318,24	2351,48
28	219,36	329,82	124,65	495,44	378,13	132,64	335,89	380,47	2396,4

Figura 46. Tabla resumen con cálculos finales

A partir de ese punto, se adecuaron los valores en diferentes archivos txt que se denominaron *dataCalculation_train.txt* y *dataCalculation_valid.txt* siguiendo el formato válido establecido por la biblioteca LIBSVM para poder posteriormente realizar su tratamiento con este entorno. En la Figura 47 se muestra un fragmento del archivo *dataCalculation_train.txt* y en la Figura 48 un fragmento del archivo *dataCalculation_valid.txt*. El número que se encuentra al comienzo de cada línea hace referencia al identificador del gesto, los siguientes números desde 1: hasta 8: hacen referencia a los valores EMG_{avg} asociados a los ocho canales del dispositivo *Myo* y, por último, el valor 9: representa el parámetro EMG_{sum}.

```
1 1:228,28 2:364,13 3:198,59 4:609,15 5:436,02 6:152,11 7:338,36 8:389,48 9:2716,12
1 1:257,58 2:304,57 3:212,42 4:587,95 5:409,65 6:148,84 7:393,35 8:505,89 9:2820,25
1 1:296,66 2:402,68 3:160,23 4:711,31 5:525,83 6:191,43 7:429,97 8:594,39 9:3312,5
1 1:358,22 2:547,87 3:195,11 4:664,57 5:500,58 6:171,31 7:398,85 8:511,74 9:3348,25
1 1:230,42 2:318,89 3:149,22 4:548,08 5:414,93 6:142,87 7:305,55 8:347,81 9:2457,77
1 1:325,15 2:506,04 3:283,98 4:617,55 5:475,11 6:185,59 7:459,3 8:581,42 9:3434,14
1 1:229,89 2:343,62 3:137,48 4:545,54 5:403,32 6:144,17 7:328,42 8:331,72 9:2464,16
1 1:287,23 2:442,65 3:179,68 4:641,43 5:465,12 6:167,68 7:362,7 8:430,45 9:2976,94
1 1:319,95 2:462,18 3:233,74 4:637,53 5:462,69 6:179,79 7:465,15 8:551,26 9:3312,29
1 1:339,38 2:512,56 3:200,26 4:608,46 5:462,07 6:177,77 7:394,43 8:502,36 9:3197,29
1 1:266,79 2:392,85 3:202,37 4:553,66 5:389,67 6:142,05 7:365,18 8:464,05 9:2776,62
1 1:185,73 2:263,01 3:120,97 4:452,78 5:303,86 6:113,65 7:283,84 8:321,45 9:2045,29
1 1:228,73 2:345,2 3:159,08 4:524,75 5:369,9 6:142,36 7:334,9 8:364,02 9:2468,94
1 1:211,92 2:302,84 3:194,41 4:486,03 5:337,12 6:121,01 7:316,67 8:361,31 9:2331,31
1 1:211,84 2:279,43 3:156,04 4:558,68 5:398,95 6:143,02 7:310,24 8:339,07 9:2397,27
1 1:224,17 2:293,98 3:174,69 4:524,53 5:368,87 6:132,87 7:349,93 8:411,37 9:2480,41
1 1:188,92 2:264,51 3:121,24 4:534,51 5:380,25 6:135,35 7:298,19 8:309,11 9:2232,08
1 1:199,96 2:306,93 3:135,21 4:493,13 5:373,64 6:135,82 7:292,88 8:340,47 9:2278,04
1 1:188,07 2:236,61 3:154,12 4:512,11 5:361,69 6:125,55 7:260,55 8:309,23 9:2147,93
1 1:196,01 2:275,17 3:144,02 4:512,76 5:368,18 6:125,21 7:303,81 8:339,44 9:2264,6
2 1:23,3 2:39,37 3:141,81 4:124,16 5:42,98 6:32,16 7:41,14 8:20,22 9:465,14
2 1:22,13 2:42,5 3:124,25 4:101,12 5:46,01 6:31,91 7:36,84 8:21,18 9:425,94
2 1:24,53 2:50,98 3:156,78 4:148,88 5:50,3 6:33,39 7:38,75 8:20,44 9:524,05
2 1:21,56 2:36,46 3:121,18 4:121,45 5:45,81 6:31,95 7:35,23 8:19,76 9:433,4
2 1:20,46 2:34,9 3:120,55 4:126,04 5:45,48 6:30,28 7:35,4 8:20,59 9:433,7
2 1:23,6 2:39,41 3:138,57 4:157,8 5:48,61 6:34,59 7:42,16 8:20,88 9:505,62
2 1:20,89 2:33,03 3:108,84 4:124,31 5:47,11 6:30,73 7:35,77 8:20,11 9:420,79
2 1:19,78 2:28,75 3:91,55 4:108,03 5:43,25 6:28,44 7:24,99 8:19,02 9:363,81
2 1:19,26 2:26,48 3:80,49 4:99,19 5:42,7 6:29,41 7:31,1 8:19,39 9:348,02
2 1:20,4 2:31,32 3:108,09 4:140,64 5:44,82 6:29,42 7:33,82 8:18,82 9:427,33
```

Figura 47. Fragmento del archivo dataCalculation_train.txt

```
1 1:234,27 2:325,94 3:171,12 4:576,27 5:402,98 6:149,14 7:341,96 8:371,11 9:2572,79
1 1:242,73 2:345,38 3:165,11 4:516 5:357,84 6:136,09 7:321,87 8:375,74 9:2460,76
1 1:196,66 2:272,74 3:186,11 4:460,64 5:336,49 6:121,7 7:297,86 8:287,84 9:2160,04
1 1:230,68 2:297,68 3:169,64 4:560,27 5:401,44 6:140,86 7:361,15 8:401,91 9:2563,63
1 1:255,27 2:346,63 3:168,81 4:646,56 5:456,05 6:157,47 7:374,13 8:436,79 9:2841,71
1 1:219,76 2:315,5 3:156,76 4:587,36 5:404,05 6:141,9 7:336,79 8:352,88 9:2515
1 1:220,13 2:308,18 3:190,24 4:519,6 5:358,24 6:125,92 7:310,93 8:318,24 9:2351,48
1 1:219,36 2:329,82 3:124,65 4:495,44 5:378,13 6:132,64 7:335,89 8:380,47 9:2396,4
2 1:19,45 2:23,84 3:51,53 4:47,4 5:38,49 6:27,67 7:23,21 8:18,85 9:250,44
2 1:19,39 2:24,59 3:63,47 4:64,32 5:43,25 6:27,86 7:23,73 8:19,92 9:286,53
2 1:19,27 2:27,84 3:80,98 4:60,28 5:42,8 6:28,71 7:23,38 8:18,1 9:301,36
2 1:20,51 2:27,76 3:84,33 4:60,69 5:40,65 6:28,54 7:24,51 8:19,26 9:306,25
2 1:19,47 2:27,99 3:84,84 4:52,16 5:40,02 6:27,32 7:22,96 8:19,21 9:293,97
2 1:20,29 2:26,75 3:83,72 4:61,73 5:42,71 6:28,99 7:23,3 8:19,56 9:307,05
2 1:19,83 2:25,19 3:54,61 4:45,43 5:41,71 6:27,79 7:22,59 8:19,43 9:256,58
2 1:19,09 2:23,02 3:52,73 4:46,6 5:41,95 6:27,04 7:23,76 8:19,12 9:253,31
3 1:469,76 2:93,75 3:120,48 4:67,95 5:73,68 6:213,04 7:227,75 8:461,03 9:1727,44
```

Figura 48. Fragmento del archivo dataCalculation_valid.txt

Una vez los datos se encontraron organizados en los archivos indicados, se pasó a utilizar el *software* LIBSVM para entrenar y validar la detección de los gestos asociados a las diferentes muestras. En la Figura 49 se presentan los comandos que se han utilizado en la ventana cmd del SO Windows. Siguiendo los pasos se tienen que escalar los datos, entrenar el modelo, e introducir valores de entrada para poder comprobarlo. Se generará un porcentaje en función del éxito en el resultado del reconocimiento.

```
svm-scale -s range -l 0 -u 10 ../data/HandGesture_train.txt > HandGesture_train_scale.txt
svm-train -g 0.0001 -c 100 HandGesture_train_scale.txt
svm-predict HandGesture_train_scale.txt HandGesture_train_scale.txt.model HandGesture_train_out.txt

svm-scale -r range -l 0 -u 10 ../data/HandGesture_valid.txt > HandGesture_valid_scale.txt
svm-predict HandGesture_valid_scale.txt HandGesture_train_scale.txt.model HandGesture_valid_out.txt
```

Figura 49. Comandos LIBSVM

En primer lugar, se escalan los datos del entrenamiento mediante el comando svm-*scale*. Con los parámetros "*-l*" (*lower*) y "*-u"* (*upper*) se establecen los límites inferior y superior de la escala, respectivamente. Se estableció inicialmente un rango comprendido entre [0, 10] para escalar los datos de entrenamiento, teniendo en cuenta que en este caso particular son valores positivos y que pueden llegar hasta 1024. Con el parámetro "*-s"* (*save*) se almacenan los factores de escala, ya que serán utilizados posteriormente para escalar los datos de validación con respecto al mismo *range*. Estos se utilizan para escalar los datos de entrenamiento, que se obtienen en el archivo *dataCalculation_train_scale.txt*. En la Figura 50 se puede consultar un fragmento de este archivo, donde se encuentran los datos escalados de salida.

```
1 1:2.81879 2:6.51429 3:2.08437 4:7.41313 5:8.13525 6:4.05844 7:7.13318 8:5.40816 9:7.75975
1 1:3.20805 2:5.37143 3:2.25806 4:7.12999 5:7.58197 6:3.92857 7:8.37472 8:7.09913 9:8.08424
1 1:3.73154 2:7.2381 3:1.6129 4:8.72587 5:9.95902 6:5.32468 7:9.18736 8:8.3965 9:9.61934
1 1:4.56376 2:10 3:2.04715 4:8.12098 5:9.44672 6:4.67532 7:8.48758 8:7.18659 9:9.73167
1 1:2.84564 2:5.6381 3:1.47643 4:6.62806 5:7.68443 6:3.73377 7:6.38826 8:4.79592 9:6.95164
1 1:4.12081 2:9.21905 3:3.13896 4:7.51609 5:8.93443 6:5.12987 7:9.86456 8:8.207 9:10
1 1:2.83221 2:6.11429 3:1.32754 4:6.58945 5:7.45902 6:3.7987 7:6.90745 8:4.56268 9:6.97348
1 1:3.61074 2:8 3:1.84864 4:7.82497 5:8.72951 6:4.54545 7:7.67494 8:6.00583 9:8.57098
1 1:4.04027 2:8.38095 3:2.51861 4:7.77349 5:8.66803 6:4.93506 7:10 8:7.76968 9:9.61934
1 1:4.30872 2:9.33333 3:2.10918 4:7.40026 5:8.66803 6:4.87013 7:8.39729 8:7.05539 9:9.26053
1 1:3.32886 2:7.04762 3:2.134 4:6.69241 5:7.17213 6:3.73377 7:7.74266 8:6.50146 9:7.94696
1 1:2.24161 2:4.59048 3:1.11663 4:5.39254 5:5.40984 6:2.79221 7:5.89165 8:4.41691 9:5.66615
```

Figura 50. Fragmento del archivo dataCalculation_train_scale.txt

En la Figura 49 se puede comprobar que el comando LIBSVM utilizado a continuación es *svm-train*. Se utiliza "*- g*" (*gamma*) para definir el valor de *gamma* en la función del *kernel*, establecido en este caso a 0'0001 y "*-c*" (*cost*) para ajustar el parámetro C de C-SVC, épsilon-SVR y nu-SVR a 100. A partir de ese punto, se genera el archivo correspondiente al modelo denominado *dataCalculation_train_scale.txt.model.* Además, se muestra en el terminal del sistema la información de los modelos que ha generado y las diferentes iteraciones que se han llevado a cabo para ello. La tercera línea ejecuta el comando *svm-predict*. Como paso previo, antes de utilizar los datos destinados para la validación, se comprueba el modelo utilizando los mismos datos de entrenamiento. Para ello, se utilizan los datos de entrenamiento escalados que se encuentran en el archivo *dataCalculation_train_scale.txt* y el modelo generado *dataCalculation_train_scale.txt.model*, generándose un archivo de salida denominado *dataCalculation_train_out.txt*. En este fichero se refleja qué gestos se han ido interpretando en cada momento, por lo que se puede comprobar qué errores se han cometido. En la Figura 51 se muestra el porcentaje de acierto que se ha obtenido. Se obtiene un porcentaje de acierto total ya que, en primer lugar, se están utilizando para validar los mismos datos con los que se ha

entrenado el modelo y, en segundo lugar, el número de muestras destinadas para la validación es bastante inferior al número de muestras de entrenamiento.

Figura 51. Resultado de la predicción con los datos de entrenamiento (Excel)

Una vez realizada esta comprobación básica, se toma el fichero donde se encuentran los datos de las muestras que se van a utilizar realmente en el proceso de validación. Si se consulta de nuevo la Figura 49 se observa que, utilizando el comando *svm-scale,* y de la misma manera que antes, se escalan los datos del archivo *dataCalculation_valid.txt* y se obtiene el archivo de salida *dataCalculation_valid_scale.txt*. En este caso, en lugar de utilizar la opción "*-s"* (*save*) para almacenar los factores de escala, se emplea "*-r*" (*restore*) para restaurar los mismos parámetros de escalado. En la Figura 52 se muestra un fragmento de los datos de validación escalados.

```
1 1:2.89933 2:5.77143 3:1.74938 4:6.98842 5:7.43852 6:3.96104 7:7.2009 8:5.14577 9:7.31045
1 1:3.00671 2:6.15238 3:1.67494 4:6.21622 5:6.51639 6:3.53896 7:6.74944 8:5.20408 9:6.961
1 1:2.38926 2:4.7619 3:1.93548 4:5.4955 5:6.08607 6:3.05195 7:6.20767 8:3.92128 9:6.02496
1 1:2.84564 2:5.2381 3:1.72457 4:6.7825 5:7.41803 6:3.66883 7:7.65237 8:5.58309 9:7.28237
1 1:3.18121 2:6.17143 3:1.71216 4:7.88932 5:8.54508 6:4.22078 7:7.94582 8:6.09329 9:8.14977
1 1:2.69799 2:5.58095 3:1.56328 4:7.12999 5:7.47951 6:3.7013 7:7.08804 8:4.8688 9:7.13261
1 1:2.71141 2:5.44762 3:1.98511 4:6.25483 5:6.53689 6:3.18182 7:6.50113 8:4.37318 9:6.6209
1 1:2.69799 2:5.84762 3:1.16625 4:5.94595 5:6.94672 6:3.40909 7:7.06546 8:5.27697 9:6.76131
2 1:0.0134228 2:0.0190476 3:0.260546 4:0.18018 5:-0.0204918 7:0.0225734 9:0.0655226
2 1:0.0134228 2:0.0380952 3:0.409429 4:0.39897 5:0.0819672 7:0.0225734 8:0.0145773 9:0.177847
2 1:0.0134228 2:0.0952381 3:0.620347 4:0.34749 5:0.0614754 6:0.0324675 7:0.0225734 9:0.224649
2 1:0.0268456 2:0.0952381 3:0.669975 4:0.34749 5:0.0204918 6:0.0324675 7:0.0451467 8:0.0145773 9:0.24025
2 1:0.0134228 2:0.0952381 3:0.669975 4:0.24453 5:0.0204918 8:0.0145773 9:0.199688
2 1:0.0268456 2:0.0761905 3:0.657568 4:0.36036 5:0.0614754 6:0.0324675 7:0.0225734 8:0.0145773 9:0.24337
```

Figura 52. Fragmento del archivo dataCalculation_valid_scale.txt

Por último, se realiza la predicción con el comando *svm-predict*. Se utilizan los datos de validación escalados que se encuentran en el archivo *dataCalculation_valid_scale.txt,* y utilizando el modelo generado anteriormente (*dataCalculation_train_scale.txt.model*), se genera un archivo de salida denominado *dataCalculation_valid_out.txt*. En la Figura 53 se comprueba que el porcentaje de acierto obtenido es de un 100%.

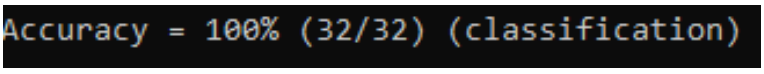

Figura 53. Resultado de la predicción con los datos de validación (Excel)

De nuevo, hay que tener en cuenta que el número de muestras destinadas para el proceso de validación es menor que el número de muestras de entrenamiento, lo que facilita la predicción. Sin embargo, se obtienen a modo de primera aproximación unos resultados positivos y se puede determinar que los parámetros calculados y utilizados para realizar las predicciones con LIBSVM pueden resultados adecuados.

4.3.2 Cálculos realizados en Python

Con la intención de realizar una doble comprobación de resultados y conseguir una manipulación más eficaz de los datos, se optó por desarrollar una segunda estrategia para procesar los datos a partir de archivos descritos en lenguaje Python. Para ello, se desarrollaron dos *scripts* que procesan de forma automática los datos experimentales que se tienen del *dataset*. Gracias a ellos, se obtienen los ficheros en el formato adecuado para el procesamiento en LIBSVM de forma más sencilla, logrando así un proceso mucho más eficaz que con el tratamiento de datos basado en Excel. Desde el script se procesan los 8 ficheros relativos a cada uno de los gestos experimentales para realizar los cálculos de los parámetros y se destinan, en base a unos porcentajes determinados, los datos para el entrenamiento y para la validación en dos archivos diferentes. Los parámetros que se ha decidido calcular han sido las medias de las cien muestras en valor absoluto de cada canal (EMG_{avg}) y, en este caso, en lugar de EMG_{sum}, la media de los valores de EMG_{avg}. Este parámetro, denominado EMG_{avg_sum}, se introdujo anteriormente planteando la fórmula correspondiente en la Figura 38.

Cabe destacar que se encontró un error en la lectura número 21 del tercer gesto *HandGesture03*, ya que contiene un número de muestras inferior a las 100 esperadas. Por tanto, para el análisis realizado a continuación se ha partido del fichero correspondiente al gesto con identificador 3 modificado. Se ha copiado la última fila de la lectura 20 a la fila defectuosa de la lectura 21. Esta repetición de los datos no se considera relevante dentro de todo el análisis del conjunto.

En primer lugar, se analiza el *script* denominado *processEMG_v01.py*, que se muestra en la Figura 54 y en la Figura 55. Para cada uno de los gestos se abre el archivo en el que se desean escribir los datos de los EMG en el formato establecido por LIBSVM. Estos se llamarán *HandGesture_0%s_proc.txt*, siendo %s el índice del gesto correspondiente. También, se abre el archivo de origen desde el que se leerán los datos, que son los correspondientes a los archivos con los datos experimentales del *dataset* (*HandGesture_0%s.txt*). Se lee cada una de las líneas del fichero de entrada y en cada una se eliminan las llaves del inicio y del final para obtener solo los valores tal y como se requieren para el formato de salida determinado por LIBSVM. Como se tienen 100 muestras, si la línea es mayor o igual que 100, se identifica esa línea como datos de las muestras. Así, se crea una nueva con los 100 primeros valores, que son los que realmente interesan. De esta manera se garantiza que únicamente se están teniendo en cuenta los valores de los EMG y no se almacenan otros caracteres como espacios en blanco, por ejemplo. Estos datos se van almacenando en una matriz que se ha llamado *sum* y, a partir de ellos, se calcula la

media de todos los valores (*mean*). Ese resultado se almacena en *data,* que es donde realmente estarán los datos que se van a volcar en los ficheros de salida. De esta forma, se consigue la media de los 100 primeros valores de la primera muestra del primer EMG. Este proceso se repite sucesivamente para cada una de las muestras.

En cambio, si la línea no es mayor o igual que 100 se comprueba si la longitud de *sum* es 8 para determinar si se han cargado ya los ocho EMG para una muestra. En ese caso, es cuando se calcula la media vertical y horizontal de los resultados. Finalmente se añade este valor (*mean_col_sum*), que corresponde a EMG_{sum}, a la matriz *data*.

```
import numpy as np
import csv
import json

for idxf in range(8):
  data = []
  sum = []
  with open("./data/HandGesture0%s_proc.txt" % (idxf + 1), "w") as fout:
    with open("./data/HandGesture0%s.txt" % (idxf + 1), "r") as fin:
      lines = csv.reader(fin)
      for idx, line in enumerate(lines):  # pylint: disable=unused-variable,redefined-outer-name
        line = [l.replace('{', '') for l in line]
        line = [l.replace('}', '') for l in line]
        emg = []
        if len(line) >= 100:
          emg.append([float(i) for i in line[0:100]])
          sum.append(emg)
          mean = np.mean(emg)
          data.append(int(mean))
        else:
          if (len(sum) == 8):
            mean_col = np.mean(sum, axis=0)
            mean_col_sum = np.mean(mean_col)
            data.append(int(mean_col_sum))
            sum = [];

    emg = 0
```

Figura 54. Código del archivo processEMG_v01.py (I)

Concretamente, en el fragmento de código que se muestra en la Figura 55, se formatea el fichero de salida, estableciendo el formato establecido por LIBSVM (1:valor 2:valor …).

```
    for idx, item in enumerate(data):  # pylint: disable=unused-variable,redefined-outer-name
      if emg == 0:  #1st mean emg value
        dic = json.dumps(item, ensure_ascii=False)
        text = "{} {}:{} ".format(str(idxf + 1), str(emg), dic)
        fout.write(text)
        emg = emg + 1
      else:
        if emg < 8: #2nd to 8th mean emg values
          dic = json.dumps(item, ensure_ascii=False)
          text = "{}:{} ".format(str(emg), dic)
          fout.write(text)
          emg = emg + 1
        else:       #mean emg_sum value
          dic = json.dumps(item, ensure_ascii=False)
          text = "{}:{}\n".format(str(emg), dic)
          fout.write(text)
          emg = 0

fin.close()
fout.close()
```

Figura 55. Código del archivo processEMG_v01.py (II)

Analizando ahora el segundo *script*, denominado *processEMG_v10.py*, se puede comparar a partir de la Figura 56 y la Figura 57, que se mantiene el código comentado anteriormente para *processEMG_v01.py*. Sin embargo, ahora se generan directamente los archivos que contienen los datos para el proceso de entrenamiento (*fout_train*) y los datos para la validación (*fout_valid*). En primer lugar, se determina qué porcentaje de datos se pretende destinar para el entrenamiento y qué porcentaje para la validación. En función de esto, se genera un número aleatorio y se comprueba si es menor o mayor que el valor de porcentaje fijado para, teniendo en cuenta esto, mezclar los datos destinándolos a un archivo o a otro. En este caso, para este fragmento del código, se ha establecido un 20% y un 80%, respectivamente. Se utiliza la misma semilla para los 8 ficheros, es decir, la secuencia de números aleatorios que se genera es la misma para cada uno de los ocho ficheros, pero de manera aleatoria.

```
with open("./data/HandGesture_valid.txt", "w") as fout_valid:
  with open("./data/HandGesture_train.txt", "w") as fout_train:
    for idxf in range(8):
      data = []
      sum = []
      with open("./data/HandGesture0%s.txt" % (idxf + 1), "r") as fin:
        lines = csv.reader(fin)
        for idx, line in enumerate(lines):  # pylint: disable=unused-variable,redefined-outer-name
          line = [l.replace('{', '') for l in line]
          line = [l.replace('}', '') for l in line]
          emg = []
          if len(line) >= 100:
            emg.append([float(i) for i in line[0:100]])
            sum.append(emg)
            mean = np.mean(emg)
            data.append(int(mean))
          else:
            if (len(sum) == 8):
              mean_col = np.mean(sum, axis=0)
              mean_col_sum = np.mean(mean_col)
              data.append(int(mean_col_sum))
              sum = [];

      emg = 0
      random.seed(5)
      if (random.random()) <= 0.2:
        file = fout_train
      else:
        file = fout_valid
```

Figura 56. Código del archivo processEMG_v10.py (I)

```
      for idx, item in enumerate(data):  # pylint: disable=unused-variable,redefined-outer-name
        if emg== 0:  #1st mean emg value
          dic = json.dumps(item, ensure_ascii=False)
          text = "{} {}:{} ".format(str(idxf + 1), str(emg + 1), dic)
          file.write(text)
          emg = emg + 1
        else:
          if emg < 8: #2nd to 8th mean emg values
            dic = json.dumps(item, ensure_ascii=False)
            text = "{}:{} ".format(str(emg + 1), dic)
            file.write(text)
            emg = emg + 1
          else:       #mean emg_sum value
            dic = json.dumps(item, ensure_ascii=False)
            text = "{}:{}\n".format(str(emg + 1), dic)
            file.write(text)
            emg = 0
            if (random.random()) < 0.2:
              file = fout_train
            else:
              file = fout_valid

fin.close()
fout_valid.close()
fout_train.close()
```

Figura 57. Código del archivo processEMG_v10.py (II)

Una vez analizado el código de Python desarrollado se pueden analizar los resultados que se han obtenido para la estrategia utilizada. Ejecutando el *scrip processEMG_v10.py* se

obtienen los archivos *handGesture_train.txt* y *handGesture_valid.txt*. En este caso se han utilizado once muestras para el entrenamiento, frente a treinta y nueve muestras para la validación. Además, se han tenido en cuenta los datos de los ocho gestos experimentales de los que se parte inicialmente, en lugar de únicamente cuatro.

Siguiendo el mismo procedimiento desarrollado anteriormente y utilizando los comandos LIBSVM correspondientes, se obtuvo, utilizando los datos de entrenamiento, un resultado del 98'8636%, como se muestra en la Figura 58. Se obtiene un acierto en la detección de 87 de las 88 muestras. Si se consulta el archivo de salida *HandGesture_train_out.txt* generado, se detecta que el error ocurre en la detección del gesto asociado con el identificador 6, ya que existe una confusión con el gesto de identificador 5.

```
Accuracy = 98.8636% (87/88) (classification)
```

Figura 58. Resultado de la predicción con los datos de entrenamiento (Python)

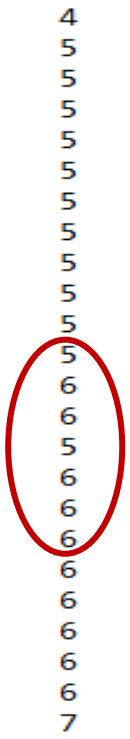

Figura 59. Comprobación de errores en el archivo HandGesture_train_out.txt

Utilizando ahora, por tanto, los datos de validación, se obtiene un porcentaje de acierto del 97'1154%. En la Figura 60 se puede comprobar que se obtiene un acierto en la detección de 303 de las 312 muestras.

```
Accuracy = 97.1154% (303/312) (classification)
```

Figura 60. Resultado de la predicción con los datos de validación (Python)

En este caso, consultando el archivo de salida *HandGesture_valid_out.txt* se detectan los errores que se muestran en la Figura 61. Los fallos se producen por confusiones entre los gestos con identificador 5 y el identificador 3 y 6, y entre el gesto con identificador 7 y el gesto con identificador 2 y 5.

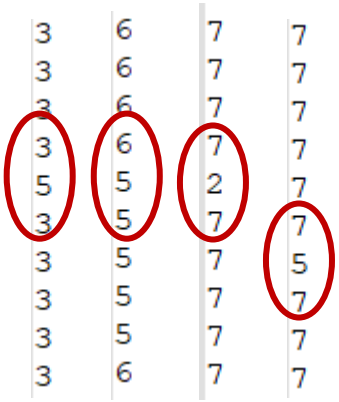

Figura 61. Comprobación de errores en el archivo HandGesture_valid_out.txt

Tras este análisis inicial se puede concluir que, aparentemente, en SVM se consiguen buenos resultados de acierto para estos símbolos con los datos de entrada experimentales. Por tanto, al incorporar todo el proceso en el dispositivo de la plataforma *hardware/software*, va a permitir que en tiempo real se puedan diferenciar los gestos. A partir de este punto, este procedimiento puede migrarse, en un siguiente paso, a los propios datos que se extraigan y obtengan mediante el uso del dispositivo *Myo Armband*.

4.3.3 Análisis cálculos realizados en Excel sin $\boldsymbol{EMG_{sum}}$

En este caso se plantea un análisis de los resultados procesados en Excel repitiendo el mismo procedimiento descrito anteriormente, pero sin tener en cuenta el cálculo del parámetro EMG_{sum}. En la Figura 62 se observa que, en este caso, el archivo *dataCalculation_train.txt* contiene únicamente los parámetros asociados a las medias de las cien muestras en valor absoluto de los ocho canales (EMG_{avg})

```
1  1 1:228,28 2:364,13 3:198,59 4:609,15 5:436,02 6:152,11 7:338,36 8:389,48
2  1 1:257,58 2:304,57 3:212,42 4:587,95 5:409,65 6:148,84 7:393,35 8:505,89
3  1 1:296,66 2:402,68 3:160,23 4:711,31 5:525,83 6:191,43 7:429,97 8:594,39
4  1 1:358,22 2:547,87 3:195,11 4:664,57 5:500,58 6:171,31 7:398,85 8:511,74
5  1 1:230,42 2:318,89 3:149,22 4:548,08 5:414,93 6:142,87 7:305,55 8:347,81
6  1 1:325,15 2:506,04 3:283,98 4:617,55 5:475,11 6:185,59 7:459,3 8:581,42
7  1 1:229,89 2:343,62 3:137,48 4:545,54 5:403,32 6:144,17 7:328,42 8:331,72
8  1 1:287,23 2:442,65 3:179,68 4:641,43 5:465,12 6:167,68 7:362,7 8:430,45
9  1 1:319,95 2:462,18 3:233,74 4:637,53 5:462,69 6:179,79 7:465,15 8:551,26
10 1 1:339,38 2:512,56 3:200,26 4:608,46 5:462,07 6:177,77 7:394,43 8:502,36
```

Figura 62. Fragmento del archivo dataCalculation_train.txt sin EMGsum

Realizando el mismo procedimiento con los comandos LIBSVM se obtiene, tanto con los datos de entrenamiento como con los de validación, un porcentaje de acierto del 100%. En la Figura 63 y en la Figura 64 se refleja cómo se han detectado correctamente las 80 muestras de entrenamiento y las 32 muestras de validación, respectivamente.

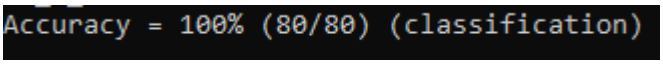

Figura 63. Resultado de la predicción con los datos de entrenamiento sin EMGsum

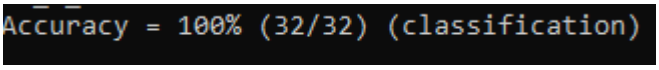

Figura 64. Resultado de la predicción con los datos de validación sin EMGsum

En definitiva, se determina que en principio no existen cambios significativos en la detección de los gestos en caso de no considerar el cálculo del parámetro EMG_{sum}.

4.3.4 Análisis cálculos realizado en Python sin $\boldsymbol{EMG_{avg\_sum}}$

De la misma manera que en el caso anterior, se plantea también el análisis de los resultados obtenidos si se procesan los datos con los *scripts* de Python, sin tener en cuenta el cálculo del parámetro EMG_{avg_sum}. En la Figura 65 se muestra el resultado obtenido al utilizar los datos de entrenamiento. Se obtiene así un porcentaje del 98'8636% al haberse detectado 87 de las 88 muestras totales analizadas.

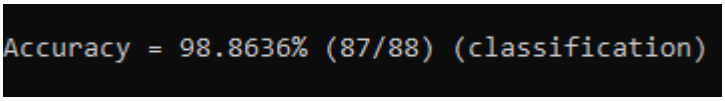

Figura 65. Resultado de la predicción con los datos de entrenamiento sin EMGsum (Python)

En el archivo *HandGesture_train_out.txt* se comprueba que se ha cometido un error en la detección del gesto con identificador 6, al confundirse con el gesto con identificador 5.

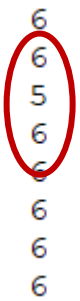

Figura 66. Comprobación de errores en el archivo HandGesture_train_out.txt

En la Figura 67 se muestra que el porcentaje obtenido tras la predicción con los datos de validación es de un 97'1154%, al haberse detectado 303 muestras de las 312 muestras totales.

```
Accuracy = 97.1154% (303/312) (classification)
```

Figura 67. Resultado de la predicción con los datos de validación sin EMGsum (Python)

De nuevo, consultando el archivo *HandGesture_valid_out.txt* se comprueba que los errores cometidos se encuentran entre los gestos con identificador 5 y los gestos con identificador 3, 6 y 8, y entre los gestos con identificador 7 y 2.

		7	
		2	
3	6	7	8
3	5	7	8
3	5	7	8
3	5	7	8
3	5	7	8
3	5	7	8
5	6	7	8
3	6	7	8
3	6	5	5
3	6	7	8
3	6	7	8

Figura 68. Comprobación de errores en el archivo HandGesture_valid_out.txt

Finalmente, para este escenario también se determina que no existen cambios relevantes en la detección de los gestos si no se tiene en cuenta el cálculo del parámetro EMG_{avg_sum}.

Por tanto, tras el análisis inicial realizado con los últimos escenarios descritos, se puede plantear descartar la utilización de los parámetros EMG_{sum} y EMG_{avg_sum} con el fin de simplificar el cómputo y conseguir un ahorro de memoria al procesar un menor número de datos.

4.3.5 Análisis cálculos realizado en Python modificando porcentajes

En este caso, se han modificado los *scripts* de Python para aumentar el porcentaje que determinaba la cantidad de datos que se destinaban para realizar el proceso de entrenamiento. Hasta ahora se había establecido un porcentaje del 20% para los datos de entrenamiento, obteniendo finalmente 39 muestras para el proceso de validación, frente a 11 muestras para el proceso de entrenamiento. En este caso, se platea la opción de aumentar el porcentaje para

conseguir menores errores. Con un mayor número de muestras destinadas para el entrenamiento, en principio el modelo podrá entrenar/aprender mejor y realizar mejores identificaciones.

En la Figura 69 se muestra el fragmento del código que se ha modificado del archivo *processEMG_v10.py*, donde se establece el nuevo porcentaje del 40%. En este caso, se obtienen 21 muestras para el entrenamiento y se destinan otras 29 para la validación.

```
if (random.random()) < 0.4:
  file = fout_train
else:
  file = fout_valid
```

Figura 69. Modificación de porcentaje en el archivo processEMG_v10.py

De nuevo, realizando el mismo procedimiento descrito anteriormente con los comandos LIBSVM, se obtiene un porcentaje de acierto del 97'619% a partir de los datos de entrenamiento, y de un 97'4138% utilizando los datos de validación.

```
Accuracy = 97.619% (164/168) (classification)
```

Figura 70. Resultado de la predicción con los datos de entrenamiento y el porcentaje modificado

En la Figura 71 se observan los errores encontrados en la detección, que se producen por algunas confusiones entre los gestos con identificador 5 y 6, y gestos con identificador 5 y 7.

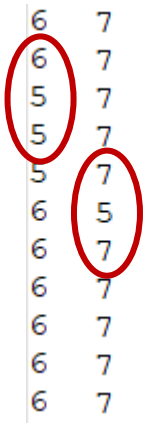

Figura 71. Comprobación de errores en el archivo HandGesture_train_out.txt

Una vez completadas las pruebas con los datos de entrenamiento, se pasa a utilizar los datos de validación. Para este caso, tal y como se muestra en la Figura 72, se obtiene un porcentaje de acierto del 97'4138%.

```
Accuracy = 97.4138% (226/232) (classification)
```

Figura 72. Resultado de la predicción con los datos de validación y el porcentaje modificado

Por último, consultando el archivo *HandGesture_valid_out.txt*, se determina que los errores en el proceso de detección se producen por las confusiones entre los gestos con identificador 5 y gestos con identificador 3, 6 y 8, y entre los gestos con identificador 2 y gestos con identificador 7. Estos errores se señalan en la Figura 73.

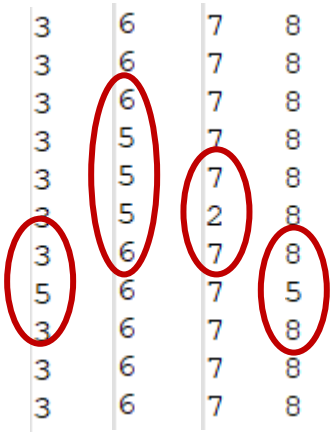

Figura 73. Comprobación de errores en el archivo HandGesture_valid_out.txt

A continuación, y a modo de resumen, se recogen los porcentajes finales obtenidos para los tres escenarios planteados en este análisis básico inicial, utilizando los *scripts* de Python que procesan los datos experimentales. Estos resultados se presentan en la Tabla 2.

Tabla 2. Resultados obtenidos con los datos procesados en Python

	Datos entrenamiento	**Datos validación**
Utilizando los cálculos de EMG_{avg} y EMG_{avg_sum}	98’8636%	97’1154%
Sin tener en cuenta el cálculo de EMG_{avg_sum}	98’8636%	97’1154%
Mayor % de datos destinados para el entrenamiento	97’619%	97’4138%

4.3.6 Conclusiones tras el análisis básico inicial

Se puede concluir, tras el análisis inicial de los diferentes escenarios planteados, y como paso previo al desarrollo de la plataforma objeto del presente TFG, que no se han conseguido

mejoras en la detección de los gestos al añadir tanto el parámetro EMG_{avg_sum} como EMG_{sum}. Por tanto, a pesar de ser parámetros que se habían considerado de interés en un inicio, no se tendrán en cuenta a la hora de procesar los datos en crudo que se obtengan a partir del dispositivo *Myo Armband*. De esta manera se ahorrarán recursos y se simplificarán los cálculos necesarios para las predicciones. Así, para los pasos posteriores, se determina que se contemplarán únicamente los parámetros correspondientes a las ocho medias de los valores absolutos de los EMG en cada instante de tiempo, los valores *pitch*, *roll* y las desviaciones estándar de la aceleración en x, y, z. En la Figura 74 se muestra un esquema del vector con los parámetros que, finalmente, se utilizarán para el proceso de clasificación.

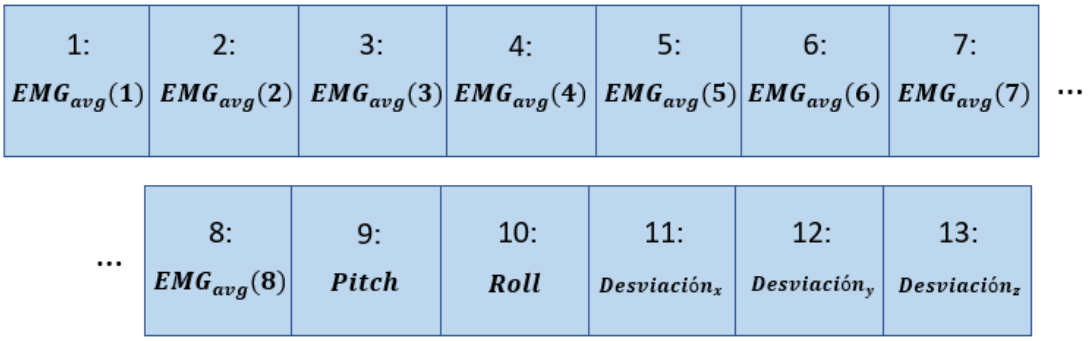

Figura 74. Vector para el proceso de clasificación

CAPÍTULO 5 Desarrollo de la plataforma *hardware/software* inicial

En este capítulo se describe el desarrollo planteado para conseguir la primera aproximación de la plataforma de bajo coste que se utilizará para el reconocimiento de gestos. En este TFG, el sistema de reconocimiento de símbolos utilizado adquiere las señales EMG a través del sensor *Myo Armband* y utiliza modelos de aprendizaje computacional o *Machine Learning* que permiten diferenciar gestos a partir de un conjunto de datos. Antes de abordar la integración completa de los procesos de obtención de datos, clasificación y detección en un único dispositivo de IoT basado en MCU, se plantea el desarrollo de una plataforma *hardware/software* inicial. El desarrollo de la plataforma se ha dividido en dos bloques iniciales para poder tener en cuenta el proceso llevado a cabo durante la implementación. Además, se presentan los aspectos más relevantes de la librería externa *myo_blelib* utilizada para una correcta comprensión del conjunto de aspectos relacionados con la comunicación inalámbrica BLE que han sido incorporados en la plataforma *hardware/software*.

5.1 Librería BLE Myo Armband

Para poder realizar una correcta conexión con el dispositivo de IoT de la familia *Arduino Nano* y el brazalete *Myo Armband* se realizará una adaptación de la librería creada en el Trabajo Fin de Máster titulado "Plataforma para la interpretación del alfabeto dactilológico de Lengua de Signos Española basada en el dispositivo *Myo Armband*" realizado por D. Carlos Santiago Viera Betancor [16]. Para el desarrollo de esta librería se tuvo en cuenta la especificación BLE del dispositivo *Myo*. Esta especificación se encuentra recogida en el archivo de cabecera C++ denominado *myohw.h* [43], proporcionado por la empresa *Thalmic Labs*. Además, se utiliza la librería *Arduino BLE*, que implementa la funcionalidad BLE en los dispositivos *Arduino* con *hardware* habilitado para ello [44]. La librería creada en el TFM indicado se divide en dos ficheros denominados *myo_blelib.h* y *myo_blelib.cpp*. El archivo de cabecera *myo_blelib.h* contiene la definición de las variables globales, la declaración de las funciones y la definición de las macros. A continuación, se realiza un análisis de su contenido y, posteriormente, al desarrollar la implementación de la plataforma, se comentará el código de las funciones de interés del fichero *myo_blelib.cpp.*

En el archivo de cabecera se encuentran los UUID (*Universally Unique Identifier*) de los servicios del dispositivo *Myo Armband*, relacionados con la información de la batería, los sensores EMG, la IMU, el clasificador interno y el servicio *Myo*. Estas definiciones se muestran en la Figura 75, y serán necesarias cuando se establezca la conexión BLE con el dispositivo *Myo.*

```
#define BatteryService              "180f" // Battery service
#define MyoserviceUUID              "d5060001-a904-deb9-4748-2c7f4a124842"
#define ControlService              "d5060001-a904-deb9-4748-2c7f4a124842" // Myo info service
#define ImuDataService              "d5060003-a904-deb9-4748-2c7f4a124842" // IMU service
#define ClassifierService           "d5060005-a904-deb9-4748-2c7f4a124842" // Classifier event service
#define EmgDataService              "d5060006-a904-deb9-4748-2c7f4a124842" // Raw EMG data service
```

Figura 75. Definición de las UUID de los servicios del Myo Armband

Asociadas a estos servicios existen una serie de características que se especifican en la Figura 76. Entre ellas se pueden destacar las características asociadas a los eventos que referencian los datos de los sensores EMG. Como el dispositivo *Myo* extrae los datos EMG a una frecuencia de 200 Hz, se necesita más de una característica BLE para realizar una transferencia. Por ello, se definen cuatro características que estén asociadas a estos datos y que se han denominado *EmgData#Characteristic*, siendo # un número comprendido entre el cero y el tres.

```
#define BatteryLevelCharacteristic    "2a19"

// Serial number for this Myo and various parameters which
// are specific to this firmware. Read-only attribute.
// See myohw_fw_info_t.
#define MyoInfoCharacteristic         "d5060101-a904-deb9-4748-2c7f4a124842"

// Current firmware version. Read-only characteristic.
// See myohw_fw_version_t.
#define FirmwareVersionCharacteristic "d5060201-a904-deb9-4748-2c7f4a124842"

// Issue commands to the Myo. Write-only characteristic.
// See myohw_command_t.
#define CommandCharacteristic         "d5060401-a904-deb9-4748-2c7f4a124842"

// // See myohw_imu_data_t. Notify-only characteristic.
#define IMUDataCharacteristic         "d5060402-a904-deb9-4748-2c7f4a124842"

// Motion event data. Indicate-only characteristic.
#define MotionEventCharacteristic     "d5060502-a904-deb9-4748-2c7f4a124842"

// Classifier event data. Indicate-only characteristic. See myohw_pose_t.
#define ClassifierEventCharacteristic "d5060103-a904-deb9-4748-2c7f4a124842"

 // Raw EMG data. Notify-only characteristic.
 #define EmgData0Characteristic       "d5060105-a904-deb9-4748-2c7f4a124842"
 #define EmgData1Characteristic       "d5060205-a904-deb9-4748-2c7f4a124842"
 #define EmgData2Characteristic       "d5060305-a904-deb9-4748-2c7f4a124842"
 #define EmgData3Characteristic       "d5060405-a904-deb9-4748-2c7f4a124842"
```

Figura 76. Definición de las UUID de las características del Myo Armband

En cuanto a los procesos de *scanning* del dispositivo *Arduino*, mostrados en la Figura 77, se debe tener en cuenta que la librería desarrollada en el TFM referenciado tenía como objetivo inicial definir una plataforma preparada para, en ocasiones futuras, desarrollar un clasificador

capaz de interpretar, además de las letras del alfabeto dactilológico de la Lengua de Signos Española, palabras y frases completas. Por eso, establecía en dos el número máximo de periféricos a los que el dispositivo de IoT podía conectarse, entendiendo la utilización de un brazalete *Myo* por cada una de las manos. Para el desarrollo del presente TFG no se contemplará en ningún caso ese escenario, por lo que se ha modificado este parámetro a 1, lo que implica el uso de un único dispositivo *Myo Armband* para el reconocimiento de gestos.

```
#define MYO_BLE_SCAN_INTERVAL          5000
#define MYO_BLE_MAX_PERIPHERALS        1
```

Figura 77. Parámetros de scanning del Myo

Para la identificación de los dispositivos, la librería declara una serie de variables globales, representadas en la Figura 78. De esta forma, se definen *MyoR* y *MyoL* para identificar el dispositivo *Myo* colocado en cada una de los antebrazos, derecho e izquierdo, respectivamente. Sin embargo, en este caso solo se tendrá en cuenta una de ellas, al no considerar la utilización de dos brazaletes. Las variables booleanas *MyoLFound* y *MyoRFound* indican si se han detectado paquetes de *advertising* de algún dispositivo *Myo,* y mediante la variable *MyoConnected* se identifica si se ha establecido la conexión. Finalmente, *MyoInfo* y *MyoFirmware* contendrán información general sobre el dispositivo *Myo*, así como la versión del *firmware* que esté utilizando. En la Figura 79 se declaran las variables globales asociadas a las características del dispositivo *Myo*. Por cada característica se declara un array de tipo *BLECharacteristic* de la librería *ArduinoBLE*.

```
//Variables globales
extern int MyoL;
extern bool MyoLFound;
extern bool MyoRFound;
extern bool MyoConnected[MYO_BLE_MAX_PERIPHERALS];
extern myohw_fw_info_t MyoInfo[MYO_BLE_MAX_PERIPHERALS];
extern myohw_fw_version_t MyoFirmware[MYO_BLE_MAX_PERIPHERALS];
```

Figura 78. Variables globales para el scanning

```
extern BLECharacteristic CommandChar[MYO_BLE_MAX_PERIPHERALS];
extern BLECharacteristic IMUDataChar[MYO_BLE_MAX_PERIPHERALS];
extern BLECharacteristic MotionEventChar[MYO_BLE_MAX_PERIPHERALS];
extern BLECharacteristic ClassifierEventDataChar[MYO_BLE_MAX_PERIPHERALS];
extern BLECharacteristic EMGData0Char[MYO_BLE_MAX_PERIPHERALS];
extern BLECharacteristic EMGData1Char[MYO_BLE_MAX_PERIPHERALS];
extern BLECharacteristic EMGData2Char[MYO_BLE_MAX_PERIPHERALS];
extern BLECharacteristic EMGData3Char[MYO_BLE_MAX_PERIPHERALS];
```

Figura 79. Características BLE locales asociadas a las características BLE del Myo

Si se analizan ahora las operaciones definidas en el archivo de cabecera, estas pueden clasificarse en cuatro grupos principales. En la Figura 80 se presenta un esquema que permite visualizar de forma gráfica y general la totalidad de las funciones definidas en la librería *myo_blelib*.

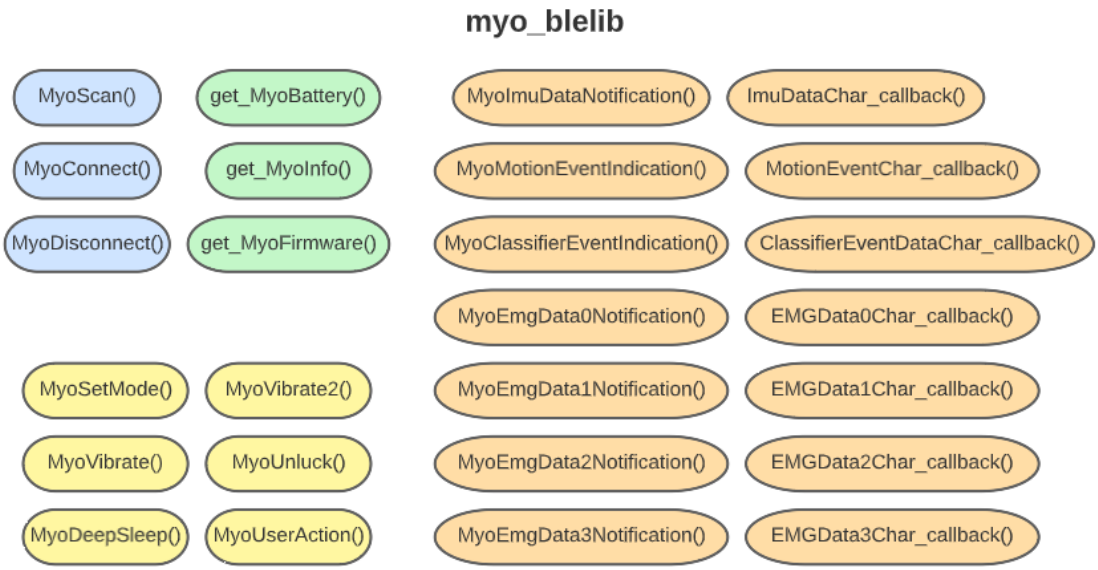

Figura 80. Funciones definidas en la librería myo_blelib

Las primeras operaciones, identificadas en color azul, están formadas por las funciones *MyoScan()* y *MyoConnect()* mostradas en la Figura 81, y están relacionadas con el proceso de inicialización. Con estas funciones se inicia el proceso de *scanning* en el dispositivo *Arduino* y se establece la conexión con el brazalete *Myo* cuando se detectan los paquetes de *advertising.* Se ha añadido, además, la función *MyoDisconnect()*, para realizar la desconexión con el dispositivo *Myo*.

```
int MyoScan(BLEDevice peripherals[]);
void MyoConnect(int MyoID, BLEDevice peripheral);
void MyoDisconnect(int MyoID, BLEDevice peripheral);
```

Figura 81. Funciones de inicialización

El segundo grupo de operaciones se ha identificado en color verde, y es el relacionado con la información básica del *Myo*. Las funciones recogidas en la Figura 82 se encargan de la obtención de datos relativos a la batería restante, así como diversa información general del dispositivo, y la versión del *firmware*.

```
void get_MyoBattery(int MyoID, BLEDevice peripheral);
void get_MyoInfo(int MyoID, BLEDevice peripheral);
void get_MyoFirmware(int MyoID, BLEDevice peripheral);
```

Figura 82. Funciones de obtención de información del dispositivo Myo

El tercer grupo de operaciones, en color naranja, hace referencia a las suscripciones de las características del brazalete, una vez establecida la conexión. Estas funciones se muestran en la Figura 83 y son necesarias para la obtención de determinados datos útiles con el fin de poder realizar el reconocimiento de los gestos de la mano. Para el desarrollo de la plataforma planteada en este TFG, únicamente será necesario la suscripción de las características *MyoImuDataNotification* y *MyoEmgData0Notification* para obtener los datos referentes a la IMU y a los sensores EMG, respectivamente. Además, en la Figura 84 se representan las llamadas de retorno ejecutadas cuando en alguna de las características suscritas como notificaciones se produce un cambio de valor.

```
void MyoImuDataNotification(int MyoID, BLEDevice peripheral, BLECharacteristic IMUDataChar, bool enable);
void MyoMotionEventIndication(int MyoID, BLEDevice peripheral, BLECharacteristic MotionEventChar, bool enable);
void MyoClassifierEventIndication(int MyoID, BLEDevice peripheral, BLECharacteristic ClassifierEventDataChar, bool enable);
void MyoEmgData0Notification(int MyoID, BLEDevice peripheral, BLECharacteristic EMGData0Char, bool enable);
void MyoEmgData1Notification(int MyoID, BLEDevice peripheral, BLECharacteristic EMGData1Char, bool enable);
void MyoEmgData2Notification(int MyoID, BLEDevice peripheral, BLECharacteristic EMGData2Char, bool enable);
void MyoEmgData3Notification(int MyoID, BLEDevice peripheral, BLECharacteristic EMGData3Char, bool enable);
```

Figura 83. Funciones para la suscripción de características del Myo

```
//void IMUDataChar_callback(int MyoID, BLECharacteristic IMUDataChar);
void MotionEventChar_callback(int MyoID, BLECharacteristic MotionEventChar);
void ClassifierEventDataChar_callback(int MyoID, BLECharacteristic ClassifierEventDataChar);
//void EMGData0Char_callback(int MyoID, BLECharacteristic EMGData0Char);
void EMGData1Char_callback(int MyoID, BLECharacteristic EMGData1Char);
void EMGData2Char_callback(int MyoID, BLECharacteristic EMGData2Char);
void EMGData3Char_callback(int MyoID, BLECharacteristic EMGData3Char);
```

Figura 84. Llamadas de retorno de las características del Myo

Finalmente, el último grupo está compuesto por el resto de las funciones para los procesos de comunicación con el brazalete y se ha identificado en color amarillo. Aunque en la Figura 85 se muestren todas ellas, en este caso solo se ha utilizado la función *MyoSetMode()*. A través de sus parámetros de entrada se indica la configuración de los modos de operación de los sensores EMG, del propio clasificador del *Myo,* y de la IMU. La definición de estos posibles modos de operación se encuentra en el archivo de cabecera *myohw.h*, del que hace uso *myo_blelib.h*, y se muestran en la Figura 86.

```
void MyoSetMode(int MyoID, BLEDevice peripheral, BLECharacteristic CommandChar, uint8_t EMG_mode,
                uint8_t IMU_mode, uint8_t Classifier_mode);
void MyoVibrate(int MyoID, BLEDevice peripheral, BLECharacteristic CommandChar, uint8_t Vibrate);
void MyoDeepSleep(int MyoID, BLEDevice peripheral, BLECharacteristic CommandChar);
void MyoVibrate2(int MyoID, BLEDevice peripheral, BLECharacteristic CommandChar,
                 uint16_t Duration0, uint8_t Stregth0, uint16_t Duration1, uint8_t Stregth1,
                 uint16_t Duration2, uint8_t Stregth2, uint16_t Duration3, uint8_t Stregth3,
                 uint16_t Duration4, uint8_t Stregth4, uint16_t Duration5, uint8_t Stregth5);
void MyoVibrate2(int MyoID, BLEDevice peripheral, BLECharacteristic CommandChar, uint8_t Sleep_mode);
void MyoUnlock(int MyoID, BLEDevice peripheral, BLECharacteristic CommandChar, uint8_t Unlock_type);
void MyoUserAction(int MyoID, BLEDevice peripheral, BLECharacteristic CommandChar, uint8_t UserAction_type);
```

Figura 85. Funciones para procesos de comunicación con el Myo

```
/// EMG modes.
typedef enum {
    myohw_emg_mode_none         = 0x00, ///< Do not send EMG data.
    myohw_emg_mode_send_emg     = 0x02, ///< Send filtered EMG data.
    myohw_emg_mode_send_emg_raw = 0x03, ///< Send raw (unfiltered) EMG data.
} myohw_emg_mode_t;

/// IMU modes.
typedef enum {
    myohw_imu_mode_none        = 0x00, ///< Do not send IMU data or events.
    myohw_imu_mode_send_data   = 0x01, ///< Send IMU data streams (accelerometer, gyroscope,
                                       ///  and orientation).
    myohw_imu_mode_send_events = 0x02, ///< Send motion events detected by the IMU
                                       /// (e.g. taps).
    myohw_imu_mode_send_all    = 0x03, ///< Send both IMU data streams and motion events.
    myohw_imu_mode_send_raw    = 0x04, ///< Send raw IMU data streams.
} myohw_imu_mode_t;

/// Classifier modes.
typedef enum {
    myohw_classifier_mode_disabled = 0x00, ///< Disable and reset the internal state of the
                                           ///  onboard classifier.
    myohw_classifier_mode_enabled  = 0x01, ///< Send classifier events (poses and arm events).
} myohw_classifier_mode_t;
```

Figura 86. Estructuras para definir los modos del EMG, IMU y clasificador del dispositivo

5.2 Bloque 1: Desarrollo de la plataforma inicial

En esta primera implementación se pretende garantizar la correcta conexión y desconexión BLE entre el dispositivo *Myo Armband* y el dispositivo de IoT, además de que es posible capturar exitosamente los datos correspondientes a los sensores EMG y el acelerómetro para el posterior entrenamiento. Con el fin de abordar el desarrollo de la plataforma inicial se plantean tres subapartados. El primero introduce los componentes *hardware* que se utilizan, el segundo desarrolla y analiza el código que implementa la funcionalidad de la plataforma, y el tercero presenta las diferentes pruebas que se realizaron para llevar a cabo la validación, y comprobar así su funcionamiento.

5.2.1 Componentes *hardware* de la plataforma inicial

Para el desarrollo de la plataforma inicial se han empleado dos componentes *hardware* principales. El primero de ellos, que actúa como dispositivo BLE central, es la placa *Arduino*. En este caso, el modelo elegido inicialmente fue el *Arduino Nano 33 IoT*. Por otra parte, como dispositivo BLE periférico, se emplea el *Myo Gesture Control Armband*. Además, se ha incluido un pulsador que se conectará a la placa *Arduino* con el fin de facilitar la secuenciación de la toma de datos para los distintos gestos. Para realizar su interconexión se conectará su alimentación Vcc al pin 3V3 del dispositivo BLE central, y la señal de salida del pulsador al pin D6, tal y como se muestra en la Figura 87. A continuación, se presentan con más detalle las características principales de los dos componentes *hardware* que conforman la plataforma inicial.

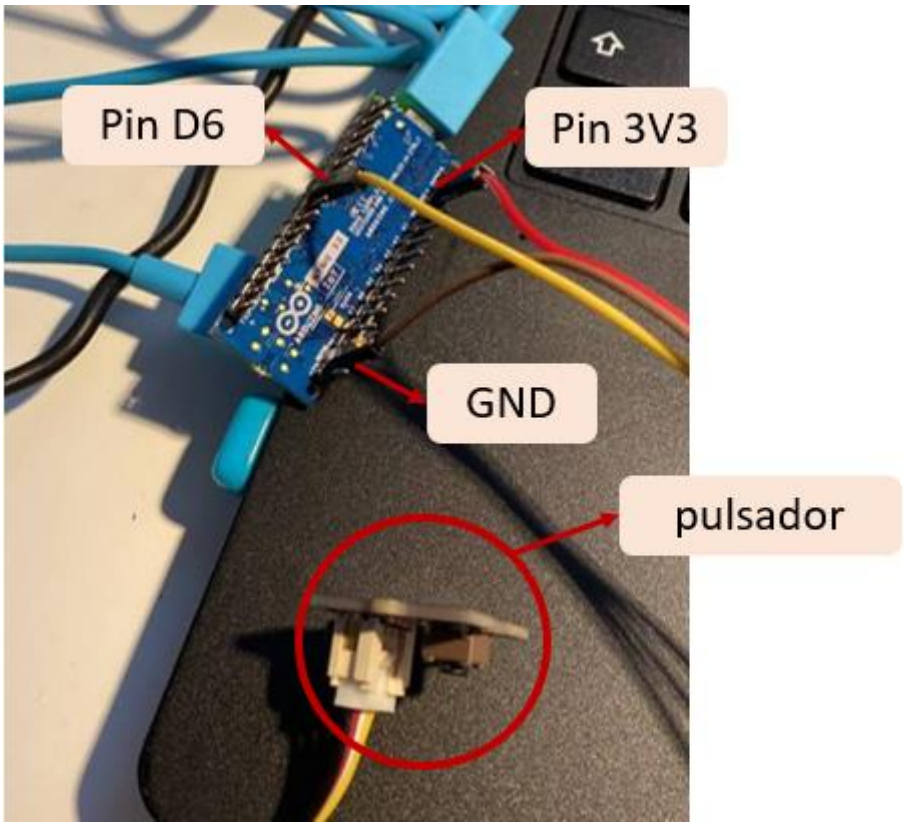

Figura 87. Conexión de componentes en la plataforma inicial

5.2.1.1 Dispositivo *Myo Gesture Control Armband*

El dispositivo *Myo Gesture Control Armband*, mostrado en la Figura 88, es un dispositivo de adquisición de señales de electromiografía (EMG). Permite la adquisición de señales y la detección de la actividad eléctrica tanto en los músculos de los brazos como en los del antebrazo gracias a su forma de brazalete, consiguiendo realizar tareas de reconocimiento de gestos. *Myo Gesture Control Armband* es un dispositivo fabricado por los laboratorios canadiense *Thalmic* que cuenta con tecnología inalámbrica por medio de comunicación Bluetooth BLE.

Dispone de un conjunto de ocho electrodos EMG o sensores de electromiografía para captar las señales eléctricas musculares y una IMU (*Inertial Measurement Unit*) de 9 ejes compuesta por un acelerómetro de 3 ejes, un giroscopio de 3 ejes y un magnetómetro de 3 ejes [5]. Además, el dispositivo está provisto de un motor de vibración que alerta al usuario cuando se producen determinados eventos. Por otra parte, dispone de dos baterías de litio recargable para alimentar el brazalete. Estas ofrecen una duración de entre 48 y 120 horas [45] en función de la aplicación que se ejecute y pueden recargarse proporcionando un valor de tensión de 5 voltios.

Figura 88. Brazalete Myo Gesture Control [46]

El funcionamiento del brazalete está gestionado por un procesador ARM Cortex M4 y la transmisión de los datos se realiza a través del chip BLE NRF51822. En la Figura 89 se muestra la placa de control electrónico del *Myo Armband*, donde se han resaltado en diferentes colores los elementos principales del dispositivo. El chip BLE NRF51822 para la transmisión inalámbrica se destaca en color azul, el microcontrolador ARM Cortex M4 en rojo, la antena que transmite los datos en color gris, el motor de vibración para la retroalimentación del usuario en naranja y el conector micro USB en violeta. Este conector se utiliza para recargar la batería y actualizar el *firmware* del dispositivo.

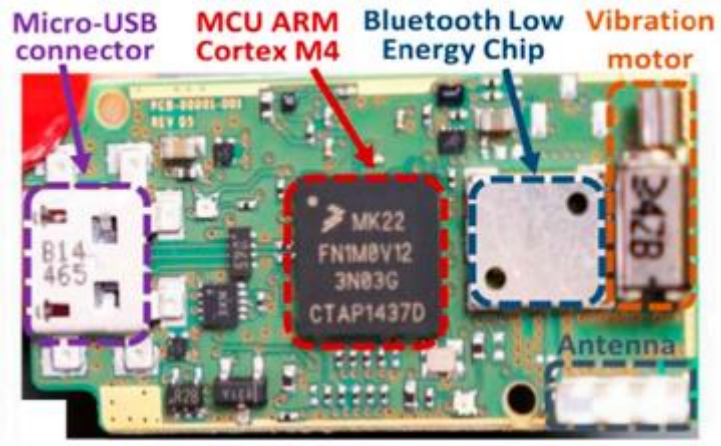

Figura 89. Vista de la placa electrónica incrustada en el elemento principal del brazalete [5]

En definitiva, *Myo* es un dispositivo inalámbrico controlado mediante gestos y movimientos. En un principio, su uso se enfocó hacia el control de aplicaciones (controlar reproducciones multimedia, navegación entre páginas, desplazamiento entre diapositivas, etc.). Por eso, el dispositivo cuenta con un clasificador interno propio que es capaz de identificar cinco gestos básicos que se representan en la Figura 90. Estos gestos son doble toque, movimiento a la derecha, movimiento a la izquierda, palma de la mano abierta, y puño, además de detectar rotaciones y movimientos gracias a la IMU.

Figura 90. Gestos definidos por defecto en el brazalete Myo [47]

A modo de resumen, en la Tabla 3 se recogen las características y especificaciones técnicas más destacables del dispositivo *Myo Armband*.

Tabla 3. Características principales del sensor Myo Armband

Dispositivo *Myo Gesture Control Armband*
Freescale Kinetis ARM Cortex M4 120 (MHz) MK22FN1M MCU
Comunicación Bluetooth con el chip BLE NRF51822

Motor de vibración
MPU-9150 9 ejes IMU (acelerómetro, giroscopio y magnetómetro en los ejes x, y, z)
8 amplificadores operacionales ST78589 (uno para cada electrodo)
2 baterías de litio de 3,7 (V) – 260 (mAh)
Frecuencia de muestreo de las señales EMG de 200 (Hz)
Frecuencia de muestreo de la unidad inercial IMU de 50 (Hz)
Retroalimentación háptica a través de vibraciones

5.2.1.2 Dispositivo *Arduino Nano* 33 *IoT*

El dispositivo Arduino Nano 33 IoT representado en la Figura 91, es una placa pequeña, robusta y potente que, como su nombre indica, tiene como objetivo su uso en proyectos relacionados con IoT (*Internet of Things*). Dispone de conectividad *WiFi* y *Bluetooth* y una arquitectura de bajo consumo que hace que constituya una solución práctica y rentable para muchos proyectos, presentando mejoras considerables de rendimiento [48].

Cuenta con un procesador ARM Cortex-M0 SAMD21 de 32 bits, un potente procesador de bajo consumo con 256 KB de memoria Flash de CPU [49]. Además, gracias al módulo WiFi uBlox NINA-W102, permite la conectividad *Bluetooth* y *WiFi* para implementar aplicaciones IoT o BLE. Dispone del criptochip ATECC608A, que almacena las claves criptográficas en *hardware*, ofreciendo un nivel muy alto de seguridad al garantizar que los datos permanezcan seguros y privados, y haciéndolo totalmente compatible con Arduino IoT Cloud [48].

Con su diseño se consiguen mejoras significativas de memoria Flash, SRAM y velocidad de reloj. Aunque su microcontrolador no permita el uso de EEPROM (*Electrically Erasable Programmable Read-Only Memory*), es posible emularlo mediante la memoria Flash reprogramable.

Figura 91. Placa Arduino Nano 33 IoT [49]

Es importante destacar que el microcontrolador del dispositivo Arduino Nano 33 IoT funciona con una tensión de alimentación de 3,3 (V), por lo que nunca se deberá aplicar un voltaje mayor a sus pines digitales y analógicos. La conexión de señales que se usan comúnmente en otras placas Arduino (como las de 5 voltios, por ejemplo) dañarán el dispositivo, por lo que se requiere un especial cuidado al conectar sensores y actuadores, asegurando que no se exceda este límite de voltaje.

A continuación, en la Tabla 4, pueden consultarse las características técnicas principales del dispositivo.

Tabla 4. Características del Arduino Nano 33 IoT

Microcontrolador	ARM Cortex-M0 SAMD21 de 32 bits
Voltaje E/S	3,3 (V)
Voltaje entrada (nominal)	5-18 (V)
Corriente DC para pines E/S	7 (mA)
Reloj	SAMD21G18A 48 (MHz)
Conector USB	Micro USB
Memoria módulo Nina W102 uBlox	256 (KB) SRAM, 1 (MB) Flash
Memoria SAMD21G18A	256 (KB) ROM, 520 (KB) SRAM, 2 (MB) Flash
Pines LEDs	13
Pines E/S digitales	14

Pines entrada analógicos	8
Pines PWM	5
Interrupciones externas	Todos los pines digitales
Módulo WiFi	Nina W102 uBlox
Módulo Bluetooth	Nina W102 uBlox
Conexión inalámbrica	WiFi 802.11b / g / ny BT 4.2
Sensores	IMU LSM6DS3
UART	RT/TX
I2C	A4 (SDA), A5 (SCL)
SPI	D11 (COPI), D12 (CIPO), D13 (SCK)
Ancho	18 (mm)
Largo	45 (mm)
Peso	5 (gr)

5.2.2 Funcionalidad de la plataforma inicial

Para abordar el *firmware* desarrollado para la plataforma inicial se presenta un diagrama de flujo global, mostrado en la Figura 92, que pretende facilitar la comprensión del código del *sketch* que se comentará en este punto de la memoria. En color amarillo se han incluido las condiciones más importantes a evaluar en el código, en color naranja las funciones incluidas en la librería *myo_blelib,* y en color azul el resto de las funciones.

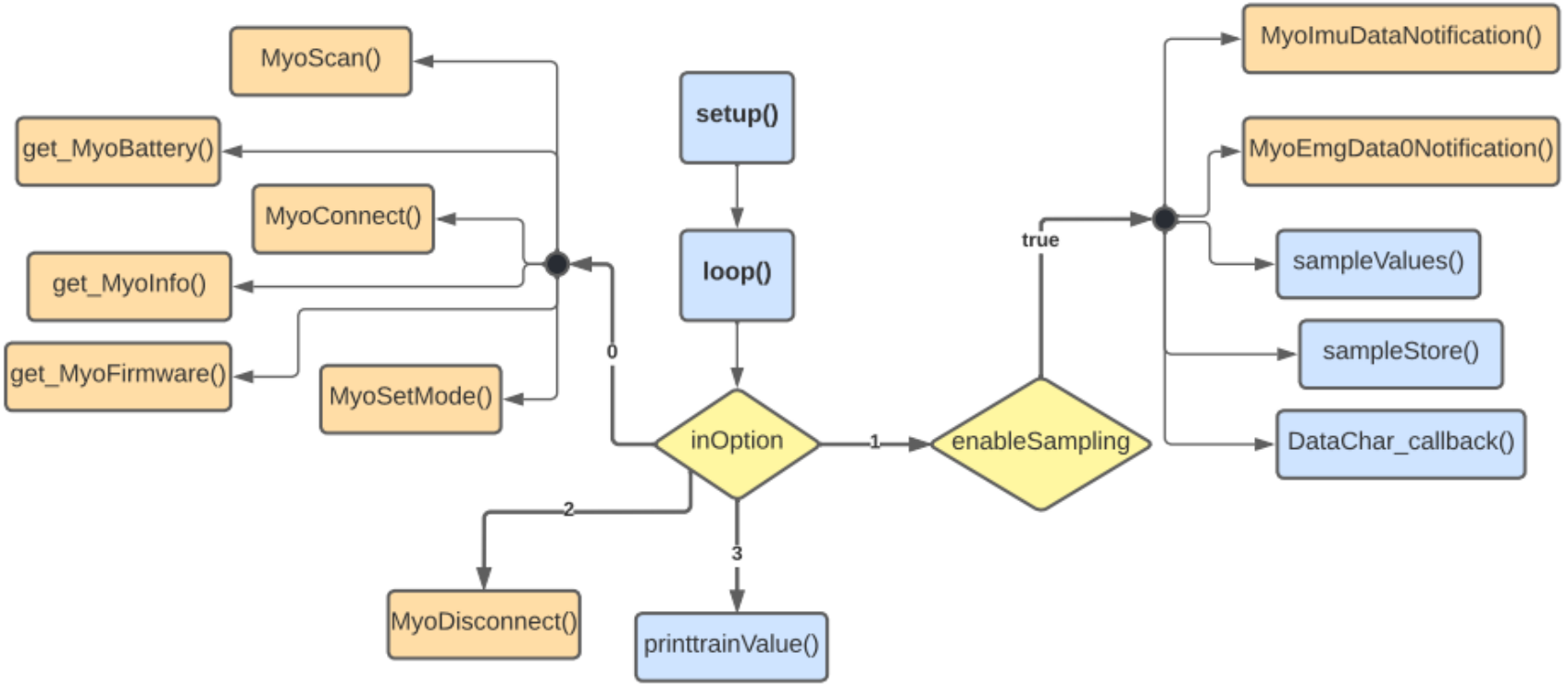

Figura 92. Diagrama de flujo del firmware de la plataforma inicial

En el esquema se observa cómo, en función de la opción elegida por el usuario en un menú inicial que se mostrará a través del terminal serie, correspondiente al valor de la variable *InOption*, se llevarán a cabo una serie de operaciones. La funcionalidad global de la plataforma inicial incorpora la comunicación con el dispositivo *Myo Armband*, permitiendo tanto la conexión como la consulta de ciertos elementos característicos del brazalete, como la batería y la suscripción a las características que se requieren para obtener las muestras correspondientes a los gestos que se realizan con la mano en cada momento.

En la Figura 93 se muestra el menú inicial planteado, con el que se inicia la ejecución del programa, incluyendo las diferentes opciones de funcionalidad de la plataforma inicial. En este caso, estarán habilitadas e implementadas cuatro opciones, correspondientes a la conexión del brazalete *Myo* (0), la captura de los datos de los gestos de la mano para el entrenamiento SVM (1), la desconexión del dispositivo *Myo* (2), y la visualización de los datos que han sido capturados (3).

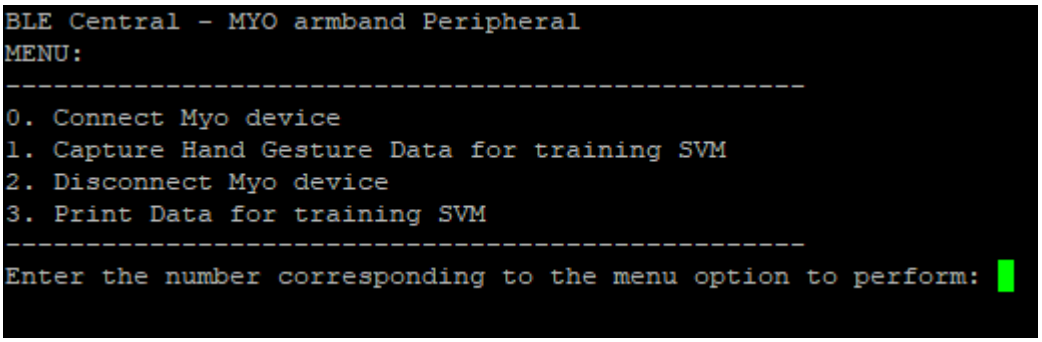

Figura 93. Menú correspondiente a la plataforma inicial

El código que muestra el menú mencionado se encuentra dentro de la función *setup()*, una de las funciones principales de la estructura básica de un *sketch Arduino*. Además, esta función prepara el programa, inicializando la comunicación serie e inicializando el hardware BLE del dispositivo *Arduino*, tal y como se puede ver en la Figura 94. La placa *Arduino* actúa como dispositivo Central en la conexión BLE con el brazalete *Myo Armband*. También, se inicializan variables globales que posteriormente, durante la ejecución del programa, modificarán su valor.

```
void setup()
{
  Serial.begin(115200);
  while (!Serial);

  pinMode(LED_BUILTIN, OUTPUT);

  // initialize the BLE hardware
  if (!BLE.begin()) {
    Serial.println("starting BLE failed!");
    while (1);
  }
  Serial.println("BLE Central - MYO armband Peripheral");
```

Figura 94. Parte del código de la función setup()

Una vez realizadas las inicializaciones en la función *setup()*, la ejecución del programa pasa a la segunda función principal, denominada *loop()*. En esta función permanecerá a la espera de que el usuario escoja una opción del menú mostrado y evaluará cada una de ellas para hacer las llamadas a las funciones correspondientes. En la Figura 95 se muestra el código asociado a la detección de la elección de la opción 0 del menú, correspondiente a la realización de la conexión con el brazalete *Myo*. En primer lugar, se inicia el proceso de búsqueda del dispositivo *Myo*, que actúa como periférico BLE. Esta búsqueda o proceso de *scanning* se lleva a cabo mediante la función *MyoScan()*, que es, durante una ventana de tiempo definida, la encargada de encontrar el dispositivo *Myo*, evaluar su nombre local y actualizar las variables *MyoLFound* y/o *MyoRFound*. Además, imprime por pantalla los eventos llevados a cabo en este método y la información sobre el dispositivo encontrado, y devuelve el número de dispositivos descubiertos, elemento que se almacena en la variable de tipo entero *MyoFound*.

Tras el proceso de *scanning,* se evalúa si se ha encontrado algún dispositivo. En este caso, como ya se comentó anteriormente, solo se tendrá en cuenta la variable asociada a un dispositivo, en concreto *MyoL*. Cuando se detecta el brazalete se procede a realizar la conexión y a descubrir los servicios y características del dispositivo periférico, mediante la función *MyoConnect()*. Por defecto, de forma inicial no se lleva a cabo la suscripción a ninguna característica, decidiéndose posteriormente qué datos leer de entre los que genera el dispositivo *Myo*. Una vez realizada la conexión, se procede a realizar la llamada a las funciones que permiten obtener la información de la batería, la información genérica del dispositivo, y la información del *firmware*. Estas son *get_MyoBattery()*, *get_MyoInfo()* y *get_MyoFirmware()*, respectivamente. Posteriormente, se ejecuta la función *MyoSetMode()*, que configura en el dispositivo *Myo* qué datos van a enviarse y cómo. Esta función debe recibir el identificador del dispositivo en el que se desee establecer el modo de actuación, el dispositivo periférico en

formato *BLEDevice*, la característica mediante la que se escribirá la configuración (*CommandChar),* y los modos de envío de los datos asociados a los sensores EMG, IMU y la habilitación del clasificador interno del dispositivo. En concreto, para este caso, se ha establecido que el modo de operación de los sensores EMG sea 0x02, para el envío filtrado de los datos EMG, que el modo de operación de la IMU sea 0x01, para el envío de datos del acelerómetro, giroscopio y orientación. Además, se ha determinado que el modo del clasificador tome el valor 0x00, para desactivar y reestablecer el estado interno del clasificador integrado.

```
  Serial.println(inOption);
  if (inOption == '0') {
  //Connect Myo device
    // start scanning for peripherals
    MyoFound = MyoScan(peripherals);

    // check if a peripheral has been discovered
    if (MyoLFound or MyoRFound) {
       if (MyoLFound) {
          MyoConnect(MyoL, peripherals[MyoL]);

          get_MyoBattery(MyoL, peripherals[MyoL]);
          get_MyoInfo(MyoL, peripherals[MyoL]);
          get_MyoFirmware(MyoL, peripherals[MyoL]);

          // Set MyoMode ------------------------------------------------------------------
          MyoSetMode(MyoL, peripherals[MyoL], CommandChar[MyoL], myohw_emg_mode_send_emg,
                     myohw_imu_mode_send_data, myohw_classifier_mode_disabled);
       }
     }
     enableSampling = false;
     Serial.println("");
     Serial.print ("Enter the number corresponding to the menu option to perform: ");
}
```

Figura 95. Código correspondiente a la opción 0 del menú

En caso de que la opción seleccionada en el menú se corresponda con el valor 1, el usuario desea capturar los datos referentes al gesto de la mano para el entrenamiento SVM. En primer lugar, se solicita que se introduzca un identificador para el gesto del cual se va a capturar la información. Mediante la función *readBytes()*, se leen los caracteres introducidos por el puerto serie y se almacenarán en el *buffer* denominado *intID[]*. Se podrán introducir hasta dos caracteres, ya que el segundo parámetro de la función indica el número de bytes a leer. Además, *readBytes()* devolverá el número de bytes almacenados en el *buffer*. El código correspondiente se representa en la Figura 96.

```
else if (inOption == '1') {
// Capture Hand Gesture Data for training SVM
   Serial.print ("Enter the id for the hand gesture data: ");
   char inID[2]={0};
   Serial.setTimeout(500);
   while (!(Serial.available() > 0)) ; // wait for serial port
   int bytesRead = Serial.readBytes(inID, 2);
   for (int i = 0; i < bytesRead; i++)
      Serial.print(inID[i]);
```

Figura 96. Código correspondiente a la opción 1 del menú (I)

Una vez se ha definido un identificador para el gesto específico se evalúa mediante una sentencia de tipo *if else*, mostrada en la Figura 97, las diferentes condiciones en función de los valores introducidos para, así, determinar en qué casos se habilita la captura de los datos. La variable booleana que permite realizar la toma de muestras se denomina *enableSampling*. Esta variable se habilita si el carácter leído es una letra comprendida entre la 'a' y la 'z', incluyendo la 'ñ' (0xC3 0xB1) y la 'ch'. Además, se almacena en la variable *symbol* el entero correspondiente al carácter introducido.

```
if ((bytesRead == 1) && (inID[0] == ' ')) {
  enableSampling = false;
  Serial.println("");
  Serial.print ("Enter the number corresponding to the menu option to perform: ");
}
else {
  if ( ((bytesRead == 1) && (((inID[0] >= 'a') && (inID[0] <= 'z')))) ||
       ((bytesRead == 2) && ((inID[0] == 0xC3) && (inID[1] == 0xB1))) ) {
    enableSampling = true;
    n_values = 0;
    symbol = (int) inID[0]; // the symbol associated to the sampled mean values
  }
  else if ((bytesRead == 2) && (((inID[0] == 'c') && (inID[1] == 'h'))
           || ((inID[0] == 'l') && (inID[1] == 'l')) )) {
    enableSampling = true;
    n_values = 0;
    symbol = (int) (inID[0] + inID[1]); // the symbol associated to the sampled mean values
  }
  else {
    enableSampling = false;
    Serial.println("");
    Serial.print ("Enter the number corresponding to the menu option to perform: ");
  }
}
```

Figura 97. Código correspondiente a la opción 1 del menú (II)

Cuando se habilita la captura de datos, se espera a que el usuario pulse el botón conectado al pin D6 del dispositivo *Arduino*. Cuando esto ocurre por primera vez, se realiza la suscripción a las características necesarias del dispositivo. Para los datos que deben capturarse en este caso, correspondientes a los valores de los sensores EMG, e IMU, es necesario suscribirse a las notificaciones de las características *MyoImuDataNotification* e *MyoEmgData0Notification*,

tal y como se muestra en el código de la Figura 98. Cuando se presentó la librería BLE *Myo Armband* se señaló la existencia de 4 características correspondientes a los eventos asociados a los datos de los sensores EMG, necesarias para poder realizar los envíos de los datos muestreados a 200 Hz. Sin embargo, para la toma de muestras de la presente plataforma, únicamente será necesario suscribirse a las notificaciones de una de esas 4 características (*MyoEmgData0Notification*). Esto es debido a que, para generar el modelo necesario para el proceso de clasificación, se necesita muestrear los datos tanto de la IMU como de los EMG a la misma frecuencia. Así, como la IMU tiene una frecuencia de refresco de 50 Hz, se obtendrá una muestra cada 20 mseg. Por tanto, se tendrá en cuenta únicamente la característica *EmgData0* (1 de cada 4), consiguiendo también así una frecuencia de 50 Hz al muestrear los datos relativos a los sensores.

Con el propósito de realizar las suscripciones a las notificaciones de las características, para cada uno de los casos, las funciones reciben como parámetro el identificador del dispositivo, el *BLEDevice* correspondiente, la característica *BLECharacteristic* a la que se desea suscribir, y un booleano que indica si se habilita o deshabilita la suscripción.

```
if (enableSampling)
{
  // Checks if BUTTONPIN is pressed to take NUMBER_SAMPLES samples
  // in order to generate NUMBER_MEAN_VALUES mean values
  bool buttonPressed = digitalRead(BUTTONPIN);
  if(buttonPressed && !buttonAlreadyPressed)
  {
    enableCaptureSampling = true;
  }
  buttonAlreadyPressed = buttonPressed;

  if (enableCaptureSampling)
  {
    if (!isSampling)
    {
      digitalWrite(LED_BUILTIN, HIGH);
      isSampling = true;
      // Subscribe to MyoImuDataNotification ------------------------------------
      MyoImuDataNotification(MyoL, peripherals[MyoL], IMUDataChar[MyoL], true);
      // Subscribe to MyoEmgDataNotification ------------------------------------
      MyoEmgData0Notification(MyoL, peripherals[MyoL], EMGData0Char[MyoL], true);
    }
```

Figura 98. Código correspondiente a la opción 1 del menú (III)

Una vez que se ha completado el proceso de suscripción a las notificaciones correspondientes, siempre que el dispositivo se mantenga conectado, y cada vez que se pulse el botón, se produce la llamada a la función *DataChar_callback()*, que atiende los cambios que se producen en las características, leyendo los nuevos valores a capturar. *DataChar_callback()*

se ha implementado realizando modificaciones en base a las funciones de la librería *myo_blelib* denominadas *IMUDataChar_callback()* y *EMGData0Char_callback()*. La llamada a esta función se muestra en la Figura 99.

```
else
{
  if (peripherals[MyoL].connected()) {
     DataChar_callback(MyoL, IMUDataChar[MyoL], EMGData0Char[MyoL]);
   }
  else {
     Serial.println("Peripheral MyoL disconnected");
  }
```

Figura 99. Código correspondiente a la opción 1 del menú (IV)

Esta función utiliza *valueUpdate()* de la librería *ArduinoBLE* para comprobar las actualizaciones de las notificaciones de las características. Si esta función retorna el valor *true* se realiza una nueva lectura mediante *readValue()*. En la Figura 100 se ve reflejada la comprobación asociada a la característica de los datos EMG y, además, se muestra cómo se almacena una muestra completa de los 8 sensores de forma local en *arrays* de tipo byte.

```
void DataChar_callback(int MyoID, BLECharacteristic IMUDataChar, BLECharacteristic EMGData0Char) {
    int8_t value[NUMBER_EMG];
    float valueacc[NUMBER_ACC];
    float accelerometer[NUMBER_MMACC];

    if (EMGData0Char.valueUpdated() && IMUDataChar.valueUpdated()) {

      // ---------------------------------------- EMG Data ------------------------------------
      myohw_emg_data_t MyoEMGData0;
      uint8_t MyoEMGData0CharSize = sizeof(myohw_emg_data_t);
      byte MyoEMGData0CharValue[MyoEMGData0CharSize];

      EMGData0Char.readValue(MyoEMGData0CharValue, MyoEMGData0CharSize);

      MyoEMGData0.sample1[0]       = (byte)MyoEMGData0CharValue[0];
      MyoEMGData0.sample1[1]       = (byte)MyoEMGData0CharValue[1];
      MyoEMGData0.sample1[2]       = (byte)MyoEMGData0CharValue[2];
      MyoEMGData0.sample1[3]       = (byte)MyoEMGData0CharValue[3];
      MyoEMGData0.sample1[4]       = (byte)MyoEMGData0CharValue[4];
      MyoEMGData0.sample1[5]       = (byte)MyoEMGData0CharValue[5];
      MyoEMGData0.sample1[6]       = (byte)MyoEMGData0CharValue[6];
      MyoEMGData0.sample1[7]       = (byte)MyoEMGData0CharValue[7];
```

Figura 100. Código de la función DataChar_callback() (I)

Con los valores almacenados en los *arrays* se procede a calcular sus valores absolutos, como se indica en la Figura 101, utilizando la función *abs()*.

```
value[0] = abs(MyoEMGData0.sample1[0]);
value[1] = abs(MyoEMGData0.sample1[1]);
value[2] = abs(MyoEMGData0.sample1[2]);
value[3] = abs(MyoEMGData0.sample1[3]);
value[4] = abs(MyoEMGData0.sample1[4]);
value[5] = abs(MyoEMGData0.sample1[5]);
value[6] = abs(MyoEMGData0.sample1[6]);
value[7] = abs(MyoEMGData0.sample1[7]);
```

Figura 101. Código de la función DataChar_callback() (II)

De la misma manera, se utiliza *readValue()* para llevar a cabo una nueva lectura de la característica asociada a los datos de la IMU, como se muestra en la Figura 102, almacenándose en los campos correspondientes los datos de orientación y los datos del acelerómetro.

```
myohw_imu_data_t MyoIMUData;
uint8_t MyoIMUDataCharSize = sizeof(myohw_imu_data_t);
byte MyoIMUDataCharValue[MyoIMUDataCharSize];

IMUDataChar.readValue(MyoIMUDataCharValue, MyoIMUDataCharSize);

MyoIMUData.orientation.w      = (byte)MyoIMUDataCharValue[1] * 256 + (byte)MyoIMUDataCharValue[0];
MyoIMUData.orientation.x      = (byte)MyoIMUDataCharValue[3] * 256 + (byte)MyoIMUDataCharValue[2];
MyoIMUData.orientation.y      = (byte)MyoIMUDataCharValue[5] * 256 + (byte)MyoIMUDataCharValue[4];
MyoIMUData.orientation.z      = (byte)MyoIMUDataCharValue[7] * 256 + (byte)MyoIMUDataCharValue[6];
MyoIMUData.accelerometer[0]   = (byte)MyoIMUDataCharValue[9] * 256 + (byte)MyoIMUDataCharValue[8];
MyoIMUData.accelerometer[1]   = (byte)MyoIMUDataCharValue[11] * 256 + (byte)MyoIMUDataCharValue[10];
MyoIMUData.accelerometer[2]   = (byte)MyoIMUDataCharValue[13] * 256 + (byte)MyoIMUDataCharValue[12];
```

Figura 102. Código de la función DataChar_callback() (III)

Los datos de orientación y los datos del acelerómetro se obtienen multiplicados por un factor de escala denominado *MYOHW_ORIENTATION_SCALE* y *MYOHW_ACCELEROMETER_SCALE*, respectivamente. Sin embargo, para realizar el cálculo del *pitch* y *roll* se necesitan los valores sin escalar. En la Figura 103, se muestra cómo los valores almacenados originalmente se convierten a variables de tipo *float,* y se dividen entre el factor de escala correspondiente. Además, los datos relativos al acelerómetro se multiplican inicialmente por un factor de 10 para que los valores se encuentren entre -160 y 160, en lugar de en un rango de ±16, facilitando así el proceso de escalado asociado a la clasificación SVM.

```
float w             = (float)MyoIMUData.orientation.w / MYOHW_ORIENTATION_SCALE;
float x             = (float)MyoIMUData.orientation.x / MYOHW_ORIENTATION_SCALE;
float y             = (float)MyoIMUData.orientation.y / MYOHW_ORIENTATION_SCALE;
float z             = (float)MyoIMUData.orientation.z / MYOHW_ORIENTATION_SCALE;
accelerometer[0]    = (float)10 * MyoIMUData.accelerometer[0] / MYOHW_ACCELEROMETER_SCALE;  // Multiplicado por 10 para que
accelerometer[1]    = (float)10 * MyoIMUData.accelerometer[1] / MYOHW_ACCELEROMETER_SCALE;  // los valores estén entre -160
accelerometer[2]    = (float)10 * MyoIMUData.accelerometer[2] / MYOHW_ACCELEROMETER_SCALE;  // y +160 (en lugar de ente +-16)
```

Figura 103. Código de la función DataChar_callback() (IV)

A partir de estos valores, tal y como se muestra en la Figura 104, se procede a calcular los valores del *pitch* y *roll*.

```
// Calculate pitch (valueacc[0]) & roll (valueacc[1])
valueacc[0] = asin(max(-1.0f, min(1.0f, 2.0f * (w * y - z * x))));
valueacc[0] = valueacc[0]*180.0/M_PI;
valueacc[1] = atan2(2.0f * (w * x + y * z), 1.0f - 2.0f * (x * x + y * y));
valueacc[1] = valueacc[1]*180.0/M_PI;
```

Figura 104. Código de la función DataChar_callback() (V)

De esta manera, los valores de los datos EMG se encontrarán almacenados en el *array* local denominado *value[]*, los valores de *pitch* y *roll* en el *array* local *valueacc[],* y los valores x, y, z del acelerómetro en el *array* local *accelerometer[]*. Para que estos valores sean accesibles de forma global, y no únicamente en la función *DataChar_callback()*, se almacenan en las matrices *mBuffer[][]*, *maccBuffer[][]* y *mmmaccBuffer[][]* para cada una de las muestras tomadas, tal y como se muestra en la Figura 105.

```
if (isSampling)
{
if (available == NUMBER_SAMPLES)
return;
  // Store EMG values in mBuffer
  for (int k = 0; k < NUMBER_EMG; k++)
    mBuffer[k][mWritePos] = value[k];

  // Store pitch & roll values in maccBuffer
  for (int k = 0; k < NUMBER_ACC; k++)
    maccBuffer[k][mWritePos] = valueacc[k];

  // Store acc x, y & z values in mmmaccBuffer
  for (int k = 0; k < NUMBER_MMACC; k++)
    mmmaccBuffer[k][mWritePos] = accelerometer[k];

mWritePos++;
```

Figura 105. Código de la función DataChar_callback() (VI)

Una vez realizada la llamada a la función *DataChar_callback()*, debe evaluarse si se han recogido todas las muestras establecidas para, en caso afirmativo, no volver a evaluar los cambios en las características hasta que vuelva a pulsarse de nuevo el botón. El número máximo de muestras por valor medio que se toma cada una de las veces que se presiona el botón se corresponde con el valor de la constante *NUMBER_SAMPLES*, mientras que el número máximo de valores medios por cada clase está determinado por el valor de la constante *NUMBER_MEAN_VALUES*, que indicará el número de veces que se toman datos por cada gesto. Cuando se ha alcanzado el número máximo *NUMBER_SAMPLES*, es decir, cuando se han capturado el número de muestras establecido para cada toma del gesto, se realiza la llamada a la función *sampleValues()*, tal y como se muestra en la Figura 106.

```
if (available == NUMBER_SAMPLES)
  {
    // print the symbol
    Serial.print(symbol);
    Serial.print(" ");

    digitalWrite(LED_BUILTIN, LOW);

    available = 0;

    sampleValues();
```

Figura 106. Código correspondiente a la opción 1 del menú (V)

Esta función es la encargada de realizar el tratamiento de los datos para conseguir los parámetros que van a conformar los vectores con los parámetros de entrenamiento. Por ello, calcula las 8 medias de los valores absolutos de cada sensor EMG, el valor medio de cada valor *pitch* y *roll*, y la desviación típica de x, y, z. Estos valores se almacenan en las variables *mean_value[]*, *mean_value_acc[]* y *stDev[]*, respectivamente. Para evitar incluir en este documento excesivos fragmentos de código y, al haber ahondado en capítulos anteriores en estos parámetros, se ha decidido no incluir la implementación de esta función. Posteriormente, se realiza la llamada a la función *sampleStore()*, que almacenará los parámetros calculados en base a los datos obtenidos en caso de que se trate de un identificador de gesto nuevo, o los reescribirá si se trata de un gesto del que ya se han almacenado los vectores anteriormente.

Por último, en términos de la captura de datos para el entrenamiento, se comprueba si se ha llegado al límite de tomas establecidas para el gesto que se está detectando, de acuerdo con el valor establecido en la constate *NUMBER_MEAN_VALUES*. Si esto ocurre, se lleva a cabo el proceso de cancelación de la suscripción de las notificaciones de las características, tal y como se muestra en la Figura 107.

```
if (n_values == NUMBER_MEAN_VALUES)
{
  // Unsubscribe to MyoImuDataNotification -------------------------------------
  MyoImuDataNotification(MyoL, peripherals[MyoL], IMUDataChar[MyoL], false);
  // Unsubscribe to MyoEmgDataNotification -------------------------------------
  MyoEmgData0Notification(MyoL, peripherals[MyoL], EMGData0Char[MyoL], false);
```

Figura 107. Código correspondiente a la opción 1 del menú (VI)

Por otra parte, para la desconexión del dispositivo *Myo* se debe elegir la opción 2 del menú inicial. En este caso, se realiza la llamada a la función *MyoDisconnect()*, implementada en *myo_blelib.cpp*, tal y como se muestra en la Figura 108, que se encarga de realizar la desconexión del dispositivo periférico indicado.

```
else if (inOption == '2') {
//Disconnect Myo device
   MyoDisconnect(MyoL, peripherals[MyoL]);
   enableSampling = false;
   Serial.println("");
   Serial.print ("Enter the number corresponding to the menu option to perform: ");
}
```

Figura 108. Código correspondiente a la opción 2 del menú

Seleccionando la opción 3 del menú inicial se consigue imprimir por pantalla los vectores que se utilizarán para el entrenamiento SVM. En la Figura 109 se muestra cómo, en este caso, se realiza la llamada a la función *printtranValues()*.

```
  else if (inOption == '3') {
    printtrainValues();
    Serial.println("");
    Serial.print ("Enter the number corresponding to the menu option to perform: ");
 }
```

Figura 109. Código correspondiente a la opción 3 del menú

La función *printtranValues()*, cuya implementación se muestra en la Figura 110, muestra los valores obtenidos a partir de la captura de datos en el formato establecido por el *software* LIBSVM. Un ejemplo del resultado obtenido tras su ejecución se presenta en la Figura 111. El primer elemento de cada línea hace referencia al número correspondiente al carácter ASCII (*American Standard Code for Information Interchange*) que se ha asociado como identificador del gesto. A continuación, se imprimen desde "1:" a "8:" los valores de las ocho medias de los valores absolutos de los EMG, seguidos de los valores de *pitch*, *roll* y las desviaciones estándar de x, y, z.

```
void printtrainValues(void) {
    if (n_train_vectors == 0) {
        Serial.println("ERROR - no training vectors");
    }
    else {
      for (int i = 0; i < n_train_vectors; i++)
      {
        Serial.print((int) prob_y[i]);
        Serial.print(" ");
        for (int j = 0; j < (NUMBER_EMG + NUMBER_ACC + NUMBER_MMACC); j++)
        {
           Serial.print(j+1);
           Serial.print(":");
           Serial.print(prob_x[i][j]);
           Serial.print(" ");
        }
        Serial.println();
      }
    }
}
```

Figura 110. Código de la función printtrainValues()

```
Enter the number corresponding to the menu option to perform: 3
97 1:-3 2:-1 3:1 4:-3 5:-1 6:1 7:-3 8:-1 9:14 10:-27 11:0 12:0 13:0
97 1:1 2:-3 3:-2 4:5 5:1 6:0 7:-5 8:1 9:15 10:-20 11:0 12:1 13:0
97 1:-1 2:-6 3:-4 4:-2 5:2 6:1 7:2 8:4 9:17 10:-19 11:0 12:1 13:0
97 1:-2 2:5 3:3 4:5 5:1 6:0 7:-2 8:-1 9:17 10:-16 11:0 12:1 13:0
97 1:5 2:4 3:6 4:1 5:4 6:2 7:-1 8:2 9:17 10:-17 11:0 12:1 13:0
98 1:6 2:-1 3:-9 4:-10 5:-3 6:-1 7:-2 8:2 9:19 10:-20 11:0 12:1 13:0
98 1:1 2:0 3:2 4:-1 5:-2 6:-1 7:-2 8:-4 9:19 10:-27 11:0 12:1 13:0
98 1:1 2:-7 3:0 4:0 5:0 6:-1 7:-1 8:0 9:16 10:-23 11:0 12:1 13:1
98 1:-2 2:1 3:-2 4:-3 5:-1 6:-1 7:-2 8:2 9:14 10:-20 11:0 12:2 13:1
98 1:-3 2:0 3:1 4:2 5:0 6:-1 7:-1 8:4 9:13 10:-16 11:1 12:1 13:1
```

Figura 111. Vectores de salida para el entrenamiento SVM

5.2.3 Validación de la plataforma inicial

Para comprobar la funcionalidad implementada en la plataforma inicial, en primer lugar se han definido cinco nuevos gestos estáticos, por lo que en este primer caso no se tendrán en cuenta los cinco últimos valores correspondientes al *pitch*, *roll* y acelerómetro, al no resultar significativos. De esta forma, el estudio se centra en los datos de los EMG, asegurando su fiabilidad, al ser la base de la toma de los datos. En la Tabla 5 se describen los gestos que se han realizado, así como su identificador asociado.

Tabla 5. Gestos estáticos

Identificador	Descripción
a (97)	Puño
b (98)	Palma de la mano abierta

c (99)	Pulgar hacia arriba
d (100)	Símbolo okey
e (101)	Símbolo victoria

Para el presente TFG se ha utilizado la aplicación *PuTTY* como terminal serie. Este herramienta, cuyo símbolo se presenta en la Figura 112, fue desarrollada por Simon Tatham para la plataforma Windows [50]. *PuTTY* es un *software* de código abierto respaldado por un grupo de voluntarios y que tiene disponible su código fuente. En el presente TFG su función principal ha sido la de establecer la comunicación serie con la placa *Arduino* para poder comprobar y analizar los datos recogidos a través del dispositivo *Myo Armband*.

Figura 112. Icono de la aplicación PuTTY

Por tanto, en primer lugar, una vez cargado el programa en la placa *Arduino*, se establece la comunicación serie gracias a este terminal. En la Figura 113, se muestra la configuración necesaria. Se ha seleccionado el COM3, puerto en el que está conectado el dispositivo BLE central, y se han establecido una velocidad de 9600 baudios, de acuerdo con la inicialización de la conexión establecida en el código de la plataforma inicial.

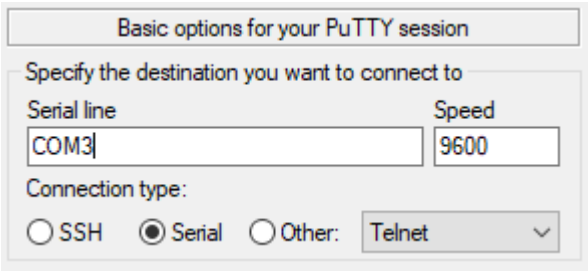

Figura 113. Establecimiento de conexión serie con el terminal PuTTY

La primera comprobación necesaria será la del establecimiento de conexión BLE con el brazalete *Myo*, seleccionando la opción 0 del menú. En la Figura 114, se muestra un fragmento de la información que se imprime por pantalla cuando se realiza la detección y conexión con el dispositivo.

```
Number of Myo peripherals found = 1 - MyoL
Connecting to Myo 0 ...
Connected to Myo 0
Discovering Myo 0 attributes ...
Attributes discovered
   * 10 Services discovered
   * 18 Characteristics discovered

---------------------------------------------------------
- MyoBattery for Peripheral 0 = 18%
---------------------------------------------------------


---------------------------------------------------------
- MyoInfo for Peripheral 0
---------------------------------------------------------
MyoInfo.serial_number = 150 66 9 104 236 214
MyoInfo.unlock_pose = myohw_pose_double_tap
MyoInfo.active_classifier_type = myohw_classifier_model_builtin
MyoInfo.active_classifier_index = 0
MyoInfo.has_custom_classifier = 1
MyoInfo.stream_indicating = 0
MyoInfo.sku = myohw_sku_unknown
```

Figura 114. Conexión con el brazalete Myo

Una vez establecida la conexión, se pueden capturar los datos de los distintos gestos estáticos. Se ha determinado, mediante el valor de las constantes definidas en el código mostrado en la Figura 115, que se obtendrán 10 tomas de 50 muestras por cada símbolo. A modo de ejemplo, en la Figura 116, se muestra la salida por pantalla de las diez primeras muestras (de las 50 que se toman cada vez que se pulsa el botón) capturadas para una de las tomas del gesto con identificador ‘a’.

```
#define NUMBER_MEAN_VALUES 10
#define NUMBER_SAMPLES 50
```

Figura 115. Número de tomas y muestras para la captura

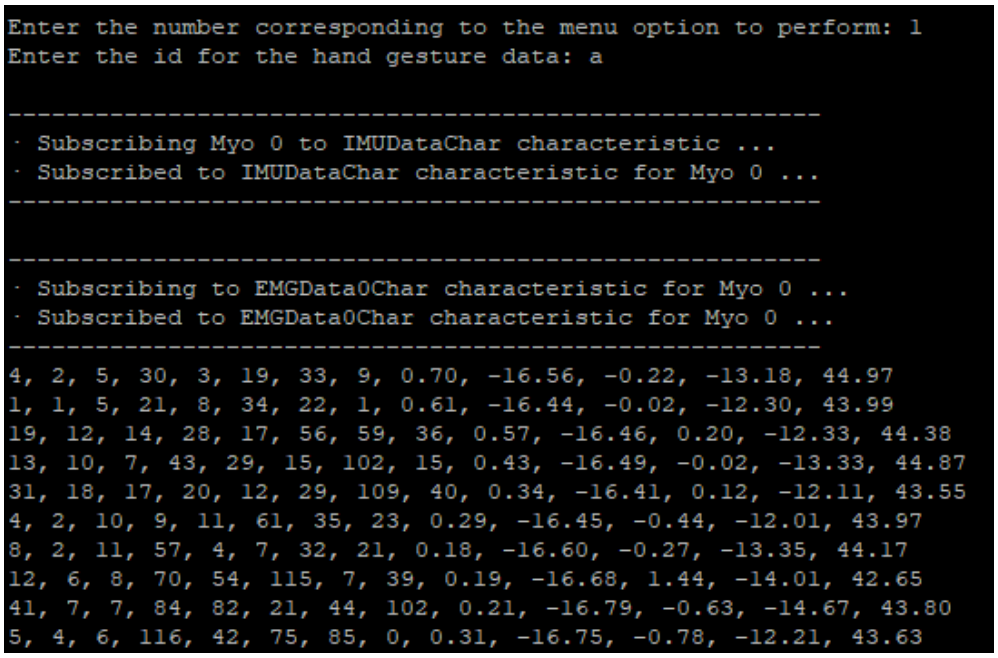

Figura 116. Captura de muestras para un gesto

Una vez que se han llevado a cabo las 10 tomas de un determinado gesto, se utiliza la opción 3 del menú, que muestra por pantalla los valores de los vectores para el entrenamiento calculados a partir de las muestras obtenidas. De esta forma, se tendrán 10 vectores con los 13 parámetros calculados precedidos del identificador 97, que es el código ASCII asociado a la letra 'a', tal y como se muestra en la Figura 117. Cabe destacar que, analizando los valores obtenidos de todos los gestos, los resultados son coherentes en cuanto a los últimos parámetros de los vectores. Los valores correspondientes a las desviaciones estándar tienen valores similares no significativos al ser gestos estáticos, mientras que los valores de *pitch* y *roll* sí que varían ligeramente entre los diferentes gestos. Este último factor es lógico ya que influye la posición de la mano y esos valores deben verse modificados. Por ejemplo, la posición de la mano no será la misma para el gesto con identificador 'd' (símbolo okey) con el de identificador 'e' (símbolo victoria). Aunque la mano permanece estática para hacer el gesto, la palma de la mano está ligeramente inclinada para el primer gesto, y no para el segundo.

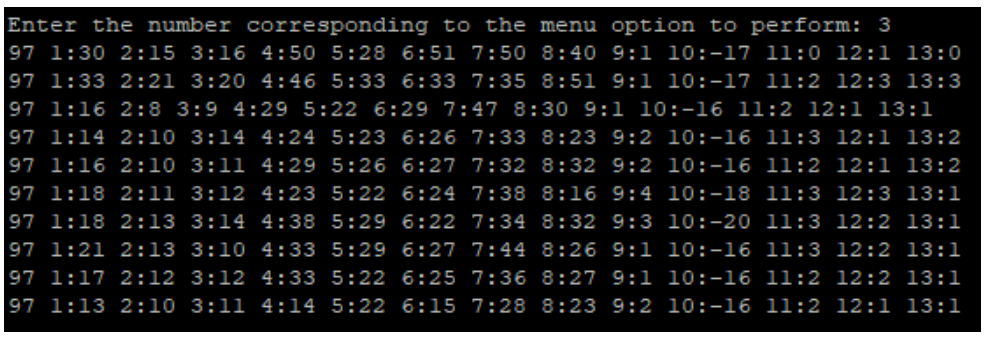

Figura 117. Vectores de entrenamiento calculados para un gesto

Este proceso se ha repetido 3 veces para cada uno de los cinco gestos estáticos, de forma que finalmente se obtuvieron 30 vectores, de los cuales, 10 se destinaron para el entrenamiento, almacenándolos en un archivo denominado *HandGesture_train,* y 20 para la validación, almacenados en un archivo denominado *HandGesture_valid*. A partir de este punto, se utilizó la herramienta externa LIBSVM para llevar a cabo el escalado, entrenamiento, generación del modelo y predicción de los gestos. Para ello, se sigue el mismo procedimiento que se detalló en el apartado 5.3 (Análisis con datos experimentales) de este documento, mostrándose únicamente los resultados obtenidos. Cuando se utilizan los mismos datos de entrenamiento para la predicción (paso previo a la utilización de los datos de validación que siempre se realizará a modo de comprobación) se obtiene un porcentaje de acierto del 98%, consiguiendo clasificar correctamente 49 de las 50 muestras, tal y como se refleja en la Figura 118. Se produce un error en la detección del gesto con identificador 'b' (98, palma de la mano abierta) al reconocerse como gesto con identificador 'e' (101, símbolo de victoria). Sin embargo, en la Figura 119 se muestra el resultado correspondiente a la utilización de los datos destinados para la validación, donde se consigue un acierto del 100% en el reconocimiento de las 100 muestras.

```
C:\Users\laura\OneDrive\Desktop\ULPGC_2\TFG\DATOS_MYO\libsvm>svm-predict HandGesture_train_scale.txt
Accuracy = 98% (49/50) (classification)
```

Figura 118. Predicción datos entrenamiento estáticos plataforma inicial

```
C:\Users\laura\OneDrive\Desktop\ULPGC_2\TFG\DATOS_MYO\libsvm>svm-predict HandGesture_valid_scale.txt
Accuracy = 100% (100/100) (classification)
```

Figura 119. Predicción datos validación estáticos plataforma inicial

Una vez garantizado que la toma de datos se realiza correctamente y que es posible llevar a cabo una predicción acertada utilizando LIBSVM, se pasó a definir otros cinco gestos dinámicos para, esta vez, considerar los parámetros de *pitch*, *roll* y desviaciones típicas en los ejes x, y, z del acelerómetro integrado en el dispositivo *Myo*. En la Tabla 6 se presentan los gestos definidos y el identificador que se les ha asociado.

Tabla 6. Gestos dinámicos

Identificador	Descripción
f (102)	Levantar el brazo (saludo)
g (103)	Supinación
h (104)	Pronación
i (105)	Flexión bíceps
j (106)	Cierre del brazo extendido hasta el pecho

La toma de datos se realiza siguiendo el mismo procedimiento descrito anteriormente y, de nuevo, se obtienen 30 vectores por cada uno de los gestos, de los cuales 10 se utilizan para el proceso de entrenamiento y 20 para el de validación. En este caso los resultados obtenidos se muestran en la Figura 120 y Figura 121, obteniendo un 100% de porcentaje de acierto.

```
C:\Users\laura\OneDrive\Desktop\ULPGC_2\TFG\DATOS_MYO\Gestos dinámicos\libsvm>svm-predict HandGesture_tra
n_out.txt
Accuracy = 100% (50/50) (classification)
```

Figura 120. Predicción datos entrenamiento dinámicos plataforma inicial

```
C:\Users\laura\OneDrive\Desktop\ULPGC_2\TFG\DATOS_MYO\Gestos dinámicos\libsvm>svm-predict HandGesture_valid
Accuracy = 100% (100/100) (classification)
```

Figura 121. Predicción datos validación dinámicos plataforma inicial

5.3 Bloque 2: Generación del modelo de forma externa

A partir de la plataforma inicial, y como paso previo a la implementación de la plataforma *hardware/software* que se pretende desarrollar en el presente TFG, se incluye una nueva funcionalidad al sistema, asociada a la habilitación de la opción número 5 del menú inicial. Esta opción permite realizar la predicción de los gestos, comprobando que se consigue identificar los símbolos de los que se recogen las muestras a partir de la generación externa del modelo, e incorporándolo en el código del *sketch* desarrollado. Para abordar este segundo bloque también se plantean tres subapartados. Un primero donde se introduce la herramienta externa *Processing* necesaria, un segundo donde se presenta y analiza el código correspondiente a la nueva funcionalidad incorporada en la plataforma, y un tercero en el que se desarrollan las comprobaciones y pruebas realizadas para la validación de su funcionamiento. En este caso, no se incluye una descripción de los componentes *hardware* porque se sigue utilizando el dispositivo *Myo Armband* para la captura de los datos y el dispositivo *Arduino Nano 33 IoT*, junto con el pulsador, para su procesamiento.

5.3.1 Herramienta *Processing*

Processing es un *software* libre y de código abierto desarrollado por Ben Fry y Casey Reas en el laboratorio de medios del Instituto de Tecnología de Massachusetts en 2001. Es un lenguaje de programación de código abierto y un entorno de desarrollo integrado creado para las comunidades de artes electrónicas y diseño visual con el objetivo de enseñar los fundamentos de la programación en un contexto más visual [51]. Utiliza lenguaje Java, aunque

hace uso de una sintaxis simplificada y un modelo de programación de gráficos. También proporciona una interfaz gráfica de usuario para simplificar la etapa de compilación y ejecución.

Figura 122. Símbolo de la herramienta Processing [51]

Esta herramienta incluye un *sketchbook*, alternativa al uso de un IDE para organizar proyectos. Cada *sketch* de *Processing* es una subclase de la clase Java PApplet, que implementa la mayoría de las características del lenguaje del *software*.

En este caso, *Processing* se utilizará para generar parte del código que se incluirá en el proyecto, y que permitirá realizar la predicción de las muestras directamente desde la plataforma. Para ello, es necesario en primer lugar haber obtenido el modelo correspondiente. Este paso, relacionado con el escalado y entrenamiento de los datos capturados, se seguirá realizando como hasta ahora, utilizando los comandos *scale* y *train* de la librería LIBSVM, a partir de los que se obtendrá como resultado un fichero con el modelo y un fichero *range* que incluye los parámetros de escalado, que serán necesarios para la ejecución del programa *Arduino_SVM* en la herramienta *Processing* [52]. Con la utilización de esta herramienta, y a partir de los archivos generados gracias a LIBSVM, se conseguirá un *sketch Arduino* de salida, del que se extraerá parte del código para utilizar en el proyecto base. Por tanto, *Arduino_SVM* permite hacer la traducción del modelo que se ha generado a partir de SVM a un código *Arduino* que puede ser fácilmente extrapolado, interpretado y reutilizado para la predicción de los gestos directamente en la plataforma.

En la Figura 123 se muestra la interfaz de usuario de la herramienta y la ventana que aparece al ejecutar el programa, donde se solicita la ruta del archivo donde se encuentra el modelo generado a partir de LIBSVM, la ruta del archivo del *range* generado, y la ruta en la que se desea almacenar la carpeta en la que se almacenarán los archivos de salida. Tras la ejecución, se generarán cuatro archivos de salida denominados *measurement.ino*, *rbf_kernel.ino*, *sketch_svm.ino* y *svm_evaluate.ino*, de los que se utilizará parte del código para la implementación de la predicción de muestras en la plataforma.

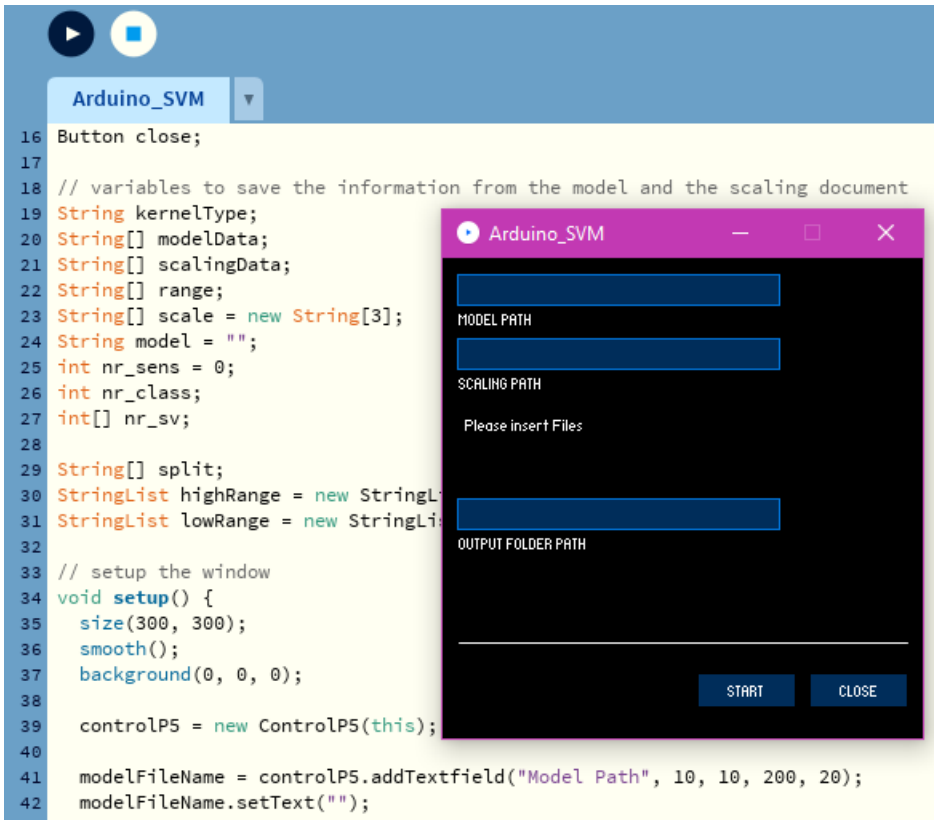

Figura 123. Programa Arduino_SVM en Processing

El archivo *Arduino_SVM* comprobará, en primer lugar, si el modelo y los parámetros de escalado indicados son válidos. En caso afirmativo, identificará si el tipo de modelo es *c_svc* y comprobará el tipo de kernel (*polynomial, linear, rbf, sigmoid*). Si todo es correcto, leerá la información de cada línea del archivo del modelo y creará el *sketch Arduino* guardando los *Support Vectors* y los parámetros *yalpha* que encuentra. Un fragmento del código que realiza este procesamiento de los datos para trasladarlos al *sketch Arduino* se muestra en la Figura 124.

```
for (int i = 0; i < modelData.length; i++) {

  split = split(modelData[i], " ");
  // if the section with SV's is already reached start to save the SV's and Yalphas
  if (sv) {

    int x = 0;
    int start = i;

    while (x < nr_sv.length) {
      String[] yalpha = new String[nr_sv.length - 1];
      String[] svs = new String[nr_sv[x]];

      for (int r = 0; r< yalpha.length; r++) {
        yalpha[r] = "";
      }
      for (int r = 0; r< svs.length; r++) {
        svs[r] = "";
      }
      for (int y = start; y < start + nr_sv[x]; y++) {
        split = split(modelData[y], " ");
        int nr = 1;

        for (int j = 0; j < yalpha.length; j++) {
          if (j == yalpha.length-1 && y == start + nr_sv[x]-1) {
            yalpha[j] = yalpha[j] +split[j];
```

Figura 124. Fragmento del código de Arduino_SVM

Finalmente, en la Figura 125 se muestra un fragmento del código que se ha obtenido del archivo de salida de *Arduino_SVM*, denominado *sketch_svm.ino*. Puede observarse cómo se definen los diferentes parámetros comentados anteriormente (*sv* y *yalpha*, entre otros) que servirán para generar los vectores que se utilizarán para comparar cada clase y realizar el sistema de votación que determinará el resultado del proceso de clasificación. Este código será el que interese reutilizar y trasladar al proyecto de la plataforma inicial desarrollado en el presente TFG.

```
#define VEC_DIM 13
#define SVM_TYPE c_svc
#define KERNEL_TYPE rbf
#define GAMMA 0.0001
#define NR_CLASS 5
#define TOTAL_SV 17
const  PROGMEM double rho[] = {-0.016243984281613458, -0.0073007111966528655, 0.04797794369193048, 0.016498579899043433,
                               0, 0.050478717531778072, 0.055991728845722004, 0.045770981670135706, 0.10780362376481128, -0.044295092037019604};
const int label[] = {102, 103, 104, 105, 106};
const int nr_sv[] = {4, 2, 2, 4, 5};
 const PROGMEM double yalpha1[4 * (NR_CLASS-1)] = {1.1084092684369251, 28.846045852491304, 10.153525934653175, 0, 29.608606446343625, 0,
                                                   0, 7.6337252495121017, 34.671962107879452, 0, 0, 0, 5.5058119300524702, 38.02964627913709, 0, 0};

 const PROGMEM double sv1[4 * VEC_DIM] = {6.66667, 5.625, 5, 10, 7.36842, 3.80952, 2, 2.94118, 1.48936, 0.289855, 7.74194, 0.416667, 9.52381,
                                          5.83333, 6.25, 6.42857, 9.04762, 8.94737, 2.38095, 2, 3.52941, 0.851064, 0.434783, 7.41935, 0.416667, 10, 6.66667,
                                          7.5, 2.85714, 10, 7.89474, 1.42857, 0.666667, 2.94118, 0.851064, 0.869565, 7.09677, 0, 10, 6.66667, 6.875, 5.71429,
                                          8.09524, 7.36842, 0.47619, 0, 0.588235, 0.638298, 1.44928, 7.09677, 0.416667, 10};

 const PROGMEM double yalpha2[2 * (NR_CLASS-1)] = {-40.107981055581412, -0, 0, 48.110713148907003, 0, 40.858834479816494, 0, 78.966989816500075};

 const PROGMEM double sv2[2 * VEC_DIM] = {6.66667, 5.625, 2.85714, 3.80952, 8.42105, 1.90476, 3.33333, 7.05882, 6.80851, 5.50725, 0.967742, 9.58333,
                                          6.66667, 5.83333, 5, 1.42857, 2.38095, 8.42105, 4.28571, 2.66667, 7.64706, 7.23404, 5.94203, 0.967742, 7.91667, 6.19048};

 const PROGMEM double yalpha3[2 * (NR_CLASS-1)] = {-22.571463072882221, -14.670868622973508, -48.110713148907003, -0, 38.856193833915633, 0, 0, 93.430990663138829};

 const PROGMEM double sv3[2 * VEC_DIM] = {3.33333, 0.625, 0, 0.952381, 0.526316, 2.85714, 5.33333, 2.35294, 4.89362, 1.15942, 2.58065, 1.25, 3.80952, 0.833333, 0.625,
```

Figura 125. Fragmento del código generado por Arduino_SVM

5.3.2 Funcionalidad de la plataforma

En este caso, el proceso de escalado y predicción de los vectores se ha implementado en esta solución, en lugar de llevarse a cabo a través de la herramienta externa LIBSVM. Para ello, se han adaptado las funciones de la librería LIBSVM encargadas de realizar las predicciones, incluyéndolas en el proyecto. De nuevo, tal y como se presentó en la plataforma inicial, se incluye en la Figura 126 el esquema actualizado correspondiente al diagrama de flujo del *sketch*. En este caso, se han señalizado en color verde las funciones que se han importado desde los archivos generados con la herramienta *Processing*.

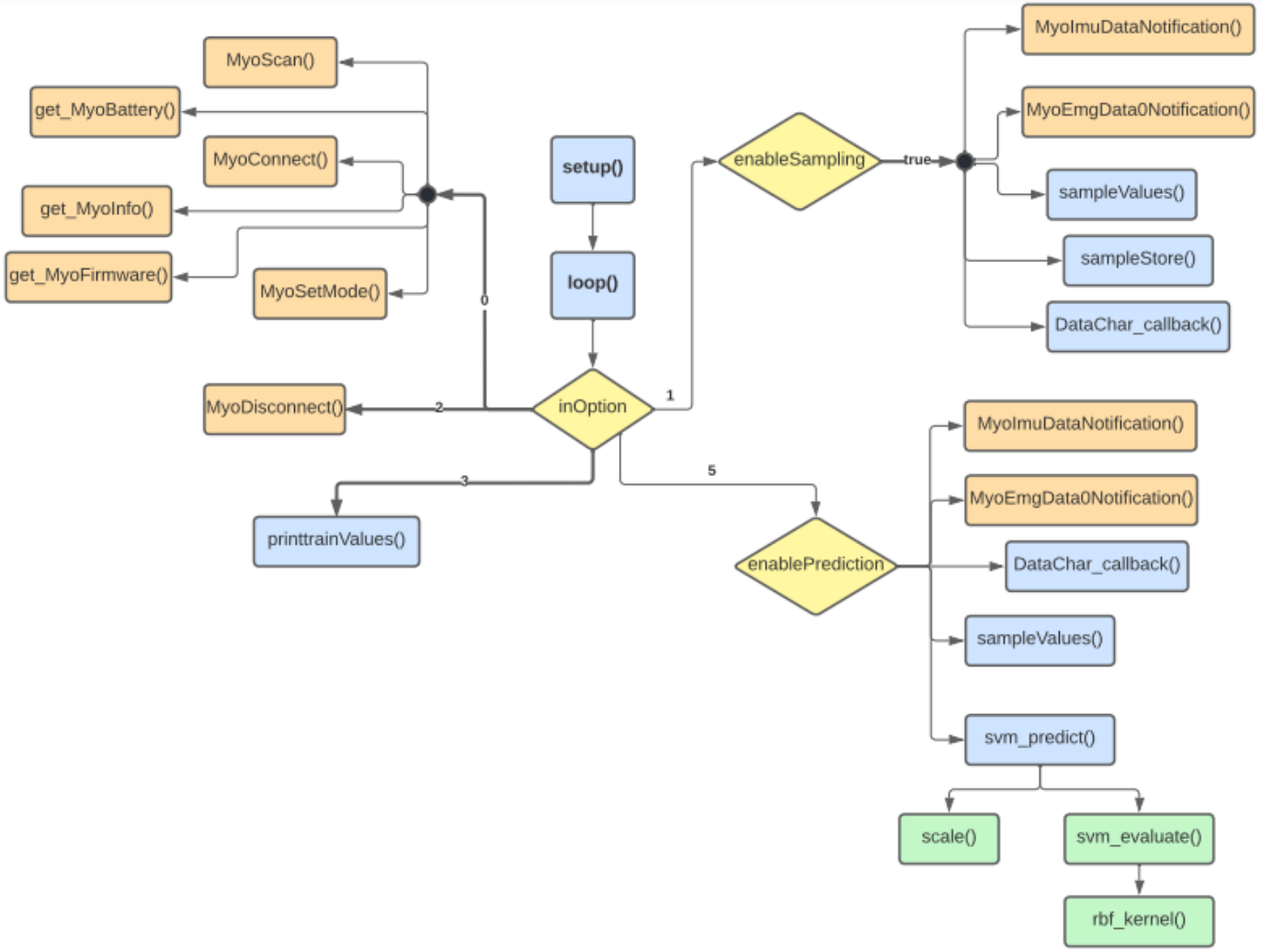

Figura 126. Diagrama de flujo del firmware de la plataforma para la predicción

De esta forma, se ha añadido funcionalidad a la opción 5 del menú inicial, antes inexistente. Habilitando esta opción se conseguirá realizar el proceso de predicción a partir de los vectores que se capturen asociados a un determinado gesto de la mano, realizando el correspondiente proceso de escalado. En la Figura 127 puede comprobarse cómo en caso de detectarse que el usuario haya elegido esta opción, la variable booleana *enablePrediction* pasa al valor *true.*

```
else if (inOption == '5') {
   enablePrediction = true;
```

Figura 127. Código correspondiente a la opción 5 del menú (I)

En este momento, el programa esperará a que el usuario presione el pulsador para iniciar el proceso de suscripción a las notificaciones de las características de la IMU y EMG del dispositivo *Myo*, tal y como se realizaba en el proceso de captura de datos, obteniéndose las muestran del gesto que el usuario realiza para la predicción. Posteriormente, se realiza la llamada a la función *sampleValues()*, encargada de calcular los parámetros que conforman los vectores formados por las ocho medias de los valores absolutos de los EMG, *picth*, *roll* y desviaciones típicas de la aceleración en x, y, z. A partir de estos valores, almacenados en *mean_value[], mean_value_acc[]* y *stDev[]* respectivamente, se conforman los vectores finales para cada toma de datos y se muestran por pantalla, tal y como se muestra en el código de la Figura 128. En este caso, no se imprime el identificador del gesto, ya que este será el elemento por predecir.

```
int sensors[NUMBER_EMG+NUMBER_ACC+NUMBER_MMACC] = {(int) round(mean_value[0]), (int) round(mean_value[1]),
                                                   (int) round(mean_value[2]), (int) round(mean_value[3]),
                                                   (int) round(mean_value[4]), (int) round(mean_value[5]),
                                                   (int) round(mean_value[6]), (int) round(mean_value[7]),
                                                   (int) round(mean_value_acc[0]), (int) round(mean_value_acc[1]),
                                                   (int) round(stDev[0]), (int) round(stDev[1]), (int) round(stDev[2])
                                                  };
for (int j = 0; j < NUMBER_EMG+NUMBER_ACC+NUMBER_MMACC; j++)
{
   Serial.print(sensors[j]);
   Serial.print(", ");
}
```

Figura 128. Código correspondiente a la opción 5 del menú (II)

Posteriormente, se llama a la nueva función *svm_predict()*, que recibe como parámetro el *array* de vectores calculado anteriormente, y que es la encargada de predecir la clase a la que pertenecen los nuevos datos medidos. Esta función realiza en primer lugar una llamada a la función *scale()*, que se ha incluido en el código pero se ha obtenido del archivo *sketch_svm.ino* generado por *Processing*, que escala los valores del *array* de vectores que recibe con ayuda de

los datos del archivo de escalado. El código asociado a la implementación de esta función se muestra en la Figura 129.

```
void scale(const int* sensor, double* scaledSensor){
   for (int p = 0; p < VEC_DIM; p++)
   {
      if (sensor[p] == low_train[p]) {
        scaledSensor[p] = (double) scaleParprob_x[0];
      }
      else if (sensor[p] == high_train[p]) {
        scaledSensor[p] = (double) scaleParprob_x[1];
      }
      else {
        scaledSensor[p] = (float)scaleParprob_x[0] +((float)scaleParprob_x[1] -((float)scaleParprob_x[0]))*
                          ((float)sensor[p]-(float)low_train[p])/((float)high_train[p]-(float)low_train[p]);
      }
   }
}
```

Figura 129. Código de la función scale()

A continuación, se compara cada clase y se comienza el proceso de votación mediante la función *svm_evaluate()*, también generada en *Processing*. Esta función evalúa si los datos proporcionados pertenecen a una clase dada. También hace uso de la función *rbf_kernel()* generada en *Processing.* En función de la votación y de los resultados obtenidos, se realiza la comprobación necesaria para determinar la clase predicha, y se muestra por pantalla el identificador del gesto correspondiente al que pertenece, tal y como se muestra en la Figura 130.

```
for(int t = 0; t < NR_CLASS; t++){
  if(result[temp] <= result[t]){
    recognizedClass = t;
    temp = t;
  }
}

Serial.println(recognizedClass, DEC);

// output of the predicted class.
Serial.print("> recognized letter = ");
if ((label[recognizedClass] >= 'a') && (label[recognizedClass] <= 'z'))
  Serial.println((char) label[recognizedClass]);
else if (label[recognizedClass] == 0xC3)
  Serial.println("ñ");
else if (label[recognizedClass] == ((int) ('c')) + ((int) ('h')))
  Serial.println("ch");
else if (label[recognizedClass] == ((int) ('l')) + ((int) ('l')))
  Serial.println("ll");
else if (label[recognizedClass] == ((int) ('r')) + ((int) ('r')))
  Serial.println("rr");


delay(500);
for(int q = 0; q < NR_CLASS; q++){
  result[q]=0;
}
```

Figura 130. Fragmento de código de la función svm_predict()

5.3.3 Validación de la plataforma

Para validar la implementación de la plataforma desarrollada, en primer lugar se realizaron comprobaciones relativas a los gestos estáticos definidos. Para ello, se tomaron como referencia las muestras relativas a las capturas que se hicieron para la validación de la plataforma inicial. A partir de los archivos correspondientes al modelo y al *range* generados utilizando la herramienta LIBSVM, se implementó el código del *sketch* necesario para incluir en el proyecto y realizar la predicción. En la Figura 131 se muestra un fragmento del código generado del archivo *sketch_svm.ino*, donde se han obtenido diferentes parámetros necesarios para la votación y evaluación de las clases que se pretenden predecir. Este código se extrae de ese archivo y se incluye en el programa que se cargará en la placa *Arduino*.

```
#define SVM_TYPE c_svc
#define KERNEL_TYPE rbf
#define GAMMA 0.0001
#define NR_CLASS 5
#define TOTAL_SV 15
const  PROGMEM double rho[] = {-0.009484104559278916 7, 0.019504513818901354, 0.00052125
const int label[] = {102, 103, 104, 105, 106};
const int nr_sv[] = {3, 4, 2, 3, 3};
 const PROGMEM double yalpha1[3 * (NR_CLASS-1)] = {2.9934967409829309, 21.2230491276404

 const PROGMEM double sv1[3 * VEC_DIM] = {2.25, 2.8, 2.22222, 8.21429, 3.95349, 3, 4.83

 const PROGMEM double yalpha2[4 * (NR_CLASS-1)] = {-24.216545868623363, -0, -0, -0, 59.

 const PROGMEM double sv2[4 * VEC_DIM] = {2, 5.6, 2.22222, 2.5, 5.34884, 3, 2.58065, 6.

 const PROGMEM double yalpha3[2 * (NR_CLASS-1)] = {-11.082026929646993, -26.31352863631

 const PROGMEM double sv3[2 * VEC_DIM] = {1.5, 1.6, 0.555556, 0.714286, 1.16279, 3.5, 2

 const PROGMEM double yalpha4[3 * (NR_CLASS-1)] = {-0.48770156494202105, -32.6878418312
```

Figura 131. Fragmento de código generado en sketch_svm.ino

Una vez se han introducido estos parámetros en el código del *sketch*, y este se ha cargado en el dispositivo *Arduino*, se selecciona la opción 5 del menú para comprobar la identificación de los gestos estáticos. Tras pulsar el pulsador, se toman las muestras necesarias y se predice la clase a la que pertenecen. En la Figura 132 se refleja el resultado obtenido por pantalla al realizar el gesto con identificador 'a' (puño). Por tanto, el resultado de la predicción se considera correcto. El proceso se repitió para los demás símbolos a clasificar, encontrando en ocasiones alguna dificultad para alguno de los gestos.

```
29, 9, 11, 6, 44, 29, 0, 52, 3.84, -16.73, -0.04, -0.24, 0.89
31, 12, 11, 47, 12, 2, -128, 64, 3.85, -16.75, -0.07, -0.29, 0.88
10, 33, 54, 70, 48, 70, 111, 33, 3.83, -16.68, -0.06, -0.26, 0.90
20, 23, 30, 106, 74, 79, 93, 32, 3.97, -16.59, -0.07, -0.29, 0.88
23, 25, 33, 31, 7, 32, 17, 9, 4.00, -16.56, -0.07, -0.24, 0.90
34, 26, 8, 46, 11, 2, 81, 38, 4.01, -16.63, -0.05, -0.26, 0.88
2, 1, 17, 32, 0, 21, 81, 59, 4.05, -16.67, -0.05, -0.25, 0.89
63, 27, 27, 71, 31, 56, 33, 28, 4.01, -16.78, -0.05, -0.29, 0.89
20, 14, 24, 84, 14, 13, 1, 21, 4.01, -16.70, -0.06, -0.27, 0.90
1:35 2:19 3:21 4:53 5:31 6:43 7:42 8:50 9:4 10:-16 11:0 12:0 13:0
· Unsubscribing to IMUDataChar characteristic for Myo 0 ...
· Unsubscribing to EMGData0Char characteristic for Myo 0 ...
35, 19, 21, 53, 31, 43, 42, 50, 4, -16, 0, 0, 0,
12.60870, 8.42105, 7.89474, 13.10345, 5.92593, 11.51515, 11.21212, 18.69565, 9.5
0000, -2.58065, 0.00000, 0.00000, 0.00000,
0
> recognized letter = a
```

Figura 132. Identificación de uno de los gestos estáticos

Para el caso de los gestos dinámicos se planteó, en primer lugar, reemplazar uno de los gestos de supinación o pronación por su elevado grado de similitud, al detectar con la validación de la plataforma inicial que podrían darse problemas para la detección entre los dos símbolos.

Por ello, se sustituye el gesto de pronación por el de dibujar un círculo en el aire. En la Tabla 7, se recogen los nuevos gestos dinámicos establecidos a partir de ahora y sus identificadores.

Tabla 7. Nuevos gestos dinámicos

Identificador	Descripción
f (102)	Levantar el brazo (saludo)
g (103)	Supinación
h (104)	Dibujar círculo
i (105)	Flexión bíceps
j (106)	Cierre del brazo extendido hasta el pecho

Además, también se quiso dar un mayor peso a los parámetros de desviación típica, por lo que, para conseguir unos valores mayores se aplicaron dos factores de multiplicación. En primer lugar, se multiplicó por un factor de 50 los valores obtenidos del acelerómetro, tal y como se muestra en la Figura 133. También, se aplicó un factor de 10 directamente al cálculo de la desviación estándar. Esta modificación se refleja en la Figura 134.

```
accelerometer[0]    = (float)50 * MyoIMUData.accelerometer[0] / MYOHW_ACCELEROMETER_SCALE;
accelerometer[1]    = (float)50 * MyoIMUData.accelerometer[1] / MYOHW_ACCELEROMETER_SCALE;
accelerometer[2]    = (float)50 * MyoIMUData.accelerometer[2] / MYOHW_ACCELEROMETER_SCALE;
```

Figura 133. Factor de escala para los valores del acelerómetro

```
Serial.print(j+NUMBER_EMG+NUMBER_ACC+1);
Serial.print(":");
Serial.print((int) round(10*(stDev[j])));
Serial.print(" ");
```

Figura 134. Factor de escala para el cálculo de la desviación típica

Como el código se ha modificado y los valores de los parámetros que conforman los vectores se calcularán de una manera diferente, se vuelven a tomar muestras para cada uno de los cinco gestos dinámicos. Se sigue el mismo procedimiento detallado anteriormente y se consiguen 20 muestras para cada uno, de las que 6 se utilizarán para el entrenamiento y 14 para la validación, distribuyéndose en dos archivos *txt* diferentes. Repitiendo el proceso de escalado, entrenamiento y predicción mediante el uso de la herramienta externa LIBSVM, se consiguió un resultado de éxito en la predicción del 100% al utilizar tanto los datos de entrenamiento como los de validación. A partir de ahí, se utiliza el modelo generado (*HandGesture_train_scale.txt.model*) y el archivo *range* para generar el archivo *sketch_svm.ino* con *Processing* y poder predecir los gestos directamente desde la plataforma. Se incorporan, de

la misma manera que para los gestos estáticos, los parámetros obtenidos a partir del modelo obtenido, y se introducen en el *firmware* de la plataforma. En la Figura 135, se muestra el resultado obtenido al utilizar la opción 5 del menú para identificar el gesto con identificador 'f' (levantar el brazo). El símbolo se identifica con éxito y, tras comprobar el resto de los símbolos dinámicos se puede determinar que se consigue un reconocimiento correcto para todos los gestos.

```
7, 2, 1, 3, 3, 4, 25, 11, 8.39, -3.88, -0.17, -0.06, 0.83
0, 0, 2, 7, 35, 1, 8, 8, 8.16, -4.01, -0.16, -0.07, 0.84
6, 1, 2, 7, 27, 12, 6, 4, 8.13, -4.15, -0.16, -0.08, 0.86
3, 0, 3, 3, 48, 16, 9, 10, 8.25, -4.15, -0.17, -0.06, 0.88
4, 1, 1, 1, 25, 5, 4, 12, 8.49, -4.09, -0.17, -0.09, 0.88
1:14 2:14 3:10 4:13 5:26 6:16 7:9 8:9 9:-33 10:-41 11:5 12:2 13:6
· Unsubscribing to IMUDataChar characteristic for Myo 0 ...
· Unsubscribing to EMGData0Char characteristic for Myo 0 ...
14, 14, 10, 13, 26, 16, 9, 9, -33, -41, 1, 0, 1,
2.50000, 4.80000, 4.44444, 2.85714, 5.11628, 3.00000, 0.32258, 2.50000, -0.52632
, -0.62500, 10.00000, 0.00000, 10.00000,
0
> recognized letter = f
```

Figura 135. Reconocimiento de uno de los gestos dinámicos

Además de estas pruebas se realizaron otras comprobaciones, introduciendo variaciones sobre los factores de escala en los gestos estáticos (agrandando las pequeñas diferencias para que sean más significativas y sea más efectivo el reconocimiento) o eliminando el cálculo de los parámetros de la desviación estándar de los gestos dinámicos para comprobar si realmente son un factor determinante en las predicciones.

CAPÍTULO 6 Desarrollo de la plataforma *hardware/software* final

En este capítulo se presenta el último paso para conseguir integrar toda la funcionalidad en la plataforma desarrollada en el presente TFG, fundamentado en la inclusión del escalado de los datos, así como de la realización del entrenamiento y la generación del modelo directamente desde la plataforma *hardware/software* final. Hasta ahora, estos puntos se habían conseguido con un procesamiento externo, gracias a la ayuda de diferentes herramientas. Para abordar el desarrollo final se plantean cuatro puntos principales. En primer lugar, los componentes *hardware* que finalmente se utilizan, planteando algunos cambios e incluyendo nuevas incorporaciones para conseguir mejoras. Un segundo punto en el que se presentan los aspectos que se han modificado en el código de la librería externa SVM para incluir las nuevas funcionalidades asociadas a la generación de modelos en la plataforma. En tercer lugar, se analiza el *firmware* desarrollado y, por último, se incluyen las pruebas realizadas para su validación.

6.1.1 Componentes *hardware* de la plataforma final

Al plantear la implementación de las nuevas funcionalidades en la plataforma surgieron los primeros problemas de memoria. Generar el modelo a partir de un conjunto de vectores de entrenamiento previamente almacenados, y almacenar el modelo generado para poder cargarlo y predecir los símbolos, supone un gasto de memoria considerable. Todo esto requiere de la utilización de punteros dinámicos, ya que no se pueden definir tamaños concretos de vectores porque no se sabe de antemano cuánto espacio va a requerir el modelo que se genera. Sin embargo, este aspecto complica la implementación de la plataforma *hardware/software* final, ya que no se dispondrá de una cantidad de memoria significativa como en el caso de un PC. Las consecuencias de este aspecto podrían derivar en no disponer del suficiente espacio y, por tanto, en una pérdida de datos. Por este motivo, se plantean dos cambios considerables respecto al *hardware*. Por un lado, se sustituirá la placa *Arduino Nano 33 IoT* por el modelo *Arduino Nano 33 BLE Sense* que integra un microcontrolador diferente. En la Tabla 8, pueden consultarse las diferencias recogidas en términos de memoria y microcontrolador entre ambos dispositivos. Además, se incorpora la memoria externa FRAM MB85RC256V, que almacenará de manera estática los vectores obtenidos para el entrenamiento, así como el modelo que se genere desde la plataforma.

Tabla 8. Comparativa entre Arduino Nano 33 IoT y Arduino Nano 33 BLE Sense

	Arduino Nano 33 IoT	*Arduino Nano 33 BLE Sense*
Microcontrolador	SAMD21G18A	nRF52840
Flash	256 KB	1 MB
SRAM/RAM	32 KB (SRAM)	256 KB (RAM)

Por su parte, el dispositivo *Myo Gesture Control Armband* seguirá cumpliendo su función como periférico BLE. Sin embargo, en la plataforma final se conectará con el nuevo dispositivo *Arduino Nano 33 BLE Sense*, que actuará como BLE central. Además, se ha incorporado una memoria externa FRAM que permitirá almacenar y cargar tanto los vectores para realizar el entrenamiento, como el modelo generado para llevar a cabo las predicciones. Por último, se incluye un *display* en el que se mostrará el resultado de las detecciones de los diferentes símbolos. La conexión entre los distintos componentes se muestra en la Figura 136.

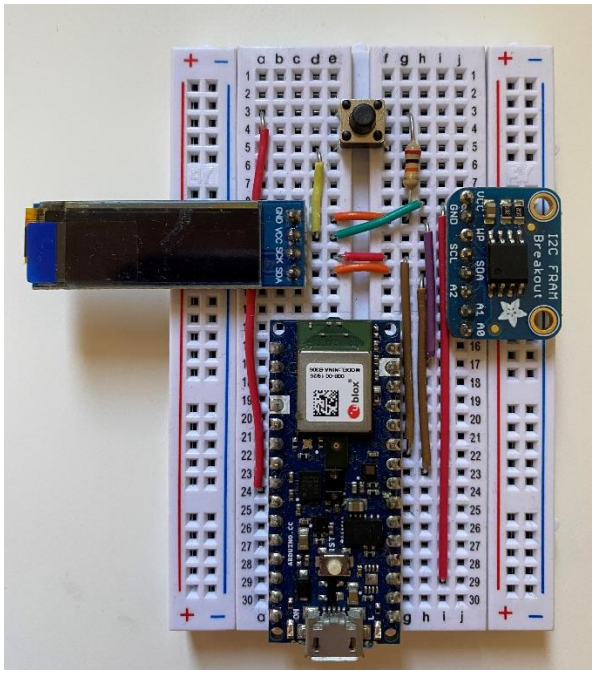

Figura 136. Conexión de componentes de la plataforma hardware/software

A continuación, se desarrolla con más detenimiento los aspectos más relevantes de los nuevos componentes *hardware* que conforman la plataforma final.

6.1.1.1 Dispositivo *Arduino Nano 33 BLE Sense*

El dispositivo *Arduino Nano 33 BLE Sense*, que se muestra en la Figura 137, es una placa que dispone de capacidad de comunicación gracias a la tecnología BLE (*Bluetooth Low Energy*) [53]. Pertenece a la familia Nano 33 y posee, entre otras cosas, un procesador diferente al de la

placa *Arduino Nano 33 IoT*. El dispositivo *Arduino Nano 33 BLE Sense* es una evolución del *Arduino Nano* tradicional, presentando un procesador mucho más potente, el nRF52840 de *Nordic Semiconductors,* con una CPU ARM Cortex-M4 de 32 bits que opera a 64 MHz [54], así como 1 MB de memoria *Flash* y 256 KB de SRAM. Esto hace que sea una placa idónea con una capacidad de procesamiento suficiente para llevar a cabo tareas de AI (*Artificial Intelligence*). Su procesador principal incluye emparejamiento *Bluetooth* a través de NFC (*Near Field Communication*) y modos de ultra bajo consumo de energía. El dispositivo *Arduino Nano 33 BLE Sense* implementa *Bluetooth* 5.0 para opciones de comunicación avanzadas y una IMU (*Inertial Measurement Unit)* de 9 ejes, entre otros sensores.

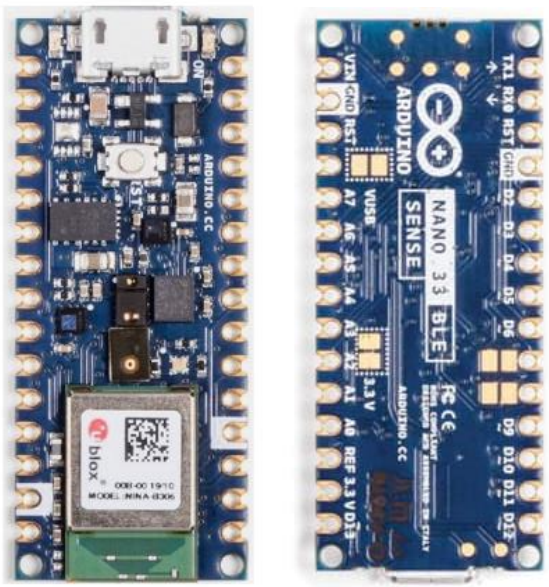

Figura 137. Placa Arduino Nano 33 BLE Sense [53]

Además, este dispositivo integra una serie de sensores útiles para realizar procesamientos de datos. En la Figura 138 se pueden identificar los componentes básicos de la placa. En color verde se señala el sensor de humedad y temperatura, que permite obtener mediciones muy precisas de las condiciones ambientales. En color naranja se indica el sensor de movimiento, vibración y orientación, que hace que el *Arduino Nano 33 BLE Sense* resulte ideal para dispositivos portátiles. Con el rectángulo rojo se señala el sensor de gestos, proximidad, color e intensidad de la luz, con el que se puede estimar la luminosidad del recinto o la proximidad de algún objeto. En color azul se indica el micrófono y en color violeta el sensor de presión. Por último, el rectángulo gris de la derecha hace referencia al microcontrolador ARM Cortex-M4 y el módulo BLE.

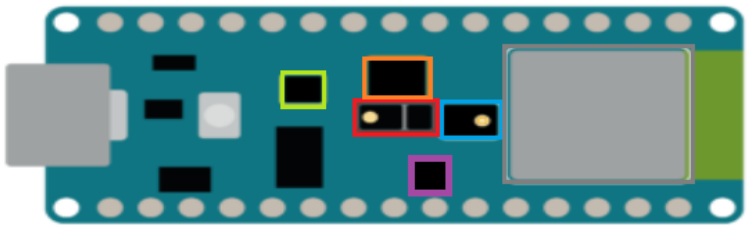

Figura 138. Visualización de los componentes del Arduino Nano 33 BLE Sense [54]

En la Tabla 9 se pueden consultar las diferentes especificaciones técnicas del dispositivo *Arduino Nano 33 BLE Sense*.

Tabla 9. Características de la placa Arduino Nano 33 BLE Sense

Microcontrolador	Nrf52840
Tensión de funcionamiento	3,3 (V)
Voltaje de entrada (límite)	21 (V)
Corriente CC por pin de E/S	15 (mA)
Velocidad de reloj	64 (MHz)
Memoria Flash de la CPU	1 (MB)
SRAM	256 (KB)
EEPROM	-
Pines de E/S digital	14
Pines PWM	Todos los pines digitales
UART	1
SPI	1
I2C	1
Pines de entrada analógica	8 (ADC 12 bits 200 kmuestras)
Pines de salida analógica	Solo a través de PWM (sin DAC)
Interrupciones externas	Todos los pines digitales
LED_BUILTIN	13
USB	Nativo en el procesador Nrf52840
IMU	LSM9DS1
Micrófono	MP34DT05
Gesto, luz, proximidad	APDS9960
Presión barométrica	LPS22HB
Temperatura, humedad	HTS221

Longitud	45 (mm)
Ancho	18 (mm)
Peso	5 (gr)

Por último, puede destacarse que cargando el programa de la plataforma *hardware/software* final en este dispositivo, el *sketch* utiliza un 34% del espacio de almacenamiento, tal y como se muestra en la Figura 139, que hace referencia a la salida obtenida tras compilar el programa. Además, las variables globales usan un 29% de la memoria dinámica, dejando suficiente espacio para variables locales. Así, aunque los requerimientos de memoria sean un aspecto primordial a tener en cuenta en el TFG, tanto para poder almacenar los vectores de entrenamiento como los modelos que se generan para la clasificación de los gestos, se cuenta con unos márgenes suficientes para garantizar el correcto almacenamiento de la información.

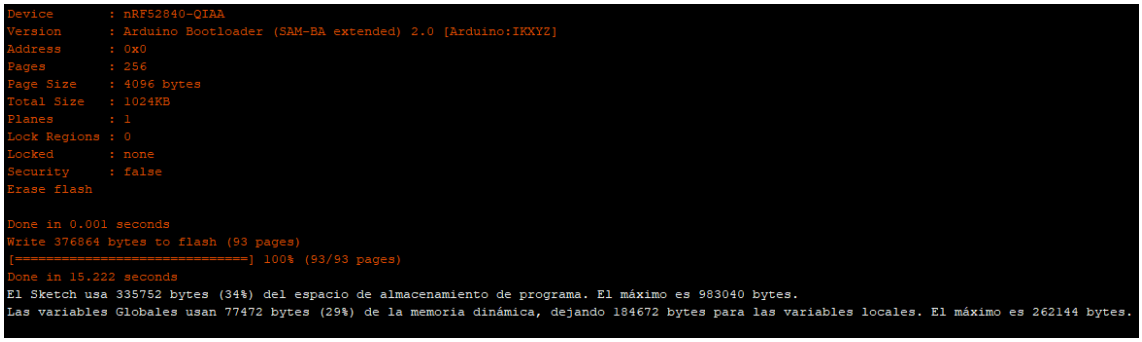

Figura 139. Disponibilidad de memoria en el dispositivo Arduino

6.1.1.2 Memoria FRAM MB85RC256V

Las memorias FRAM (*Ferroelectric Random Access Memory*) son memorias RAM (*Random Access Memory*) ferroeléctricas no volátiles, ya que son capaces de almacenar los datos incluso cuando no dispone de alimentación. Utilizan una capa ferroeléctrica en lugar de una dieléctrica para alterar un campo eléctrico a su alrededor y lograr así la no volatilidad. Esta película delgada de material ferroeléctrico suele estar fabricada de titanato de circonato de plomo [55]. Los átomos de esta capa cambian de polaridad en un campo eléctrico, consiguiendo que la película se comporte como un interruptor binario. De esta forma se consigue almacenar los datos de manera eficiente. Si no se dispone de alimentación, como la polaridad de la capa ferroeléctrica no sufre alteraciones, la información se mantiene intacta.

Si se compara con otro tipo de memorias como las DRAM (*Dynamic Random Access Memory*) o memorias de solo lectura programable y borrable eléctricamente, la memoria FRAM consume 3 mil veces menos energía y se estima que dura 10 mil veces más [56]. Esto se debe a la capacidad que presenta para escribir, borrar y reescribir en multitid de ocasiones. Además,

permite realizar escrituras mucho más rápido que otras memorias, ya que cuentan con un tiempo de acceso reducido que hace que puedan funcionar como memoria principal con la mayoría de microprocesadores. Su proceso de lectura es destructivo, por lo que se requiere una arquitectura de escritura tras la lectura.

Consumen poca energía cuando se escribe o reescribe información en el chip, por lo que se suelen utilizar para tarjetes inteligentes y dispositivos móviles que requieren de un buen rendimiento de energía.

El chip MB85RC256V, mostrado en la Figura 140, es una memoria FRAM de 256 Kbits (32 Kbytes) de almacenamiento con interfaz serie I2C (*Inter-Integrated Circuit*), que utiliza las tecnologías de proceso ferroeléctrico y de proceso CMOS (*Complementary Metal Oxide Semiconductor*) para formar las células de memoria no volátil. Dado que la FRAM es capaz de escribir a alta velocidad a pesar de ser una memoria no volátil es adecuada para la gestión de registros, almacenamiento de datos de reestablecimientos, etc. Puede funcionar a velocidades de I2C de hasta 1 MHz [57]. Cada byte puede leerse y escribirse instantáneamente un número aproximado de 10.000.000.000 de veces, por lo que su desgaste no es una preocupación. En definitiva, este chip posee una combinación de las mejores prestaciones de la memoria SRAM (*Static Random Access Memory*) y *Flash*, permitiendo almacenar datos con una alta velocidad y sin necesidad de preocuparse por la pérdida de información.

Figura 140. Memoria FRAM MB85RC256V [57]

6.1.1.3 *Display* OLED 0.91”

Este dispositivo se corresponde con una pequeña pantalla idónea para aplicaciones portátiles donde se requiere priorizar el mínimo tamaño y un bajo consumo de energía [58]. Este módulo OLED (*Organic Light-Emitting Diode*) de 0.91 pulgadas ofrece una resolución de 129*32 píxeles, permitiendo controlar cada píxel individualmente. La pantalla, mostrada en la Figura 141, es muy compacta y permite la conexión con Arduino a través de la inferfaz serie I2C. El módulo está diseñado para trabajar directamente a 5V gracias a su regulador de voltaje en placa

y puede trabajar con sistemas de 3.3V o 5V sin necesidad de conversores [59]. Presenta buen brillo y contraste, y permite mostrar texto, mapas de bits, píxeles, rectángulos, círculos o líneas. Para manejar el *display* es necesario utilizar un microcontrolador de al menos 1K de RAM que se utilizará a modo de *buffer*.

Figura 141. Display OLED 0.91" [58]

Las pantallas basadas en la tecnología OLED presentan menor grosor y peso, al no necesitar iluminación trasera, y están diseñadas para tener un ampio ángulo visual. Además, consumen mucha menos energía que la tecnología LED o LCD (*Liquid Crystal Display*), tienen tiempo de reacción rápido y son menos susceptibles al calor y al frío. Su proceso de fabricación es fácil, re requieren pocos materiales y proporcionan una mejor calidad visual. Sin embargo, una de las desventajas que presentan es que su tiempo de vida es corta (20 mil horas aproximadamente) y sus píxeles pueden deteriorarse fácilmente.

Por tanto, las pantallas OLED destacan por su gran contraste y mínimo consumo de energía [60]. Esto es debido a que cada píxel genera su luz y no necesita retroalimentación como en el caso de la tecnología LCD. Actualmente se utilizan para la fabricación de pantallas finas y flexibles destinadas a la tecnología del papel electrónico, aunque, debido a su tamaño, capacidad de visualización y demás propiedades, suele utilizarse también en relojes inteligentes, MP3 o dispositivos de salud portátiles, entre otros.

6.1.2 Adaptación del código LIBSVM

Como se ha comentado anteriormente, ha sido necesario realizar una modificación del código de LIBSVM para adaptarlo al dispositivo *Arduino* y poder integrar las nuevas funcionalidades en la plataforma. Una de estas nuevas funcionalidades es la del entrenamiento

y generación del modelo a partir de las muestras obtenidas. Para poder implementar la función *svm-train* de la librería LIBSVM en el dispositivo *Arduino* es necesario, en primer lugar, analizar el código C correspondiente a la implementación del paquete software LIBSVM, estudiando el archivo *svm.cpp*. Es un archivo denso que presenta múltiples dependencias entre las funciones que lo conforman. Por ello, es importante identificar las funciones realmente útiles para las tareas de entrenamiento, descartando aquellas destinadas a *kernels* distintos al RBF y los casos para métodos de clasificación que no se utilizan en la plataforma desarrollada en este TFG. Adicionalmente, fue necesario analizar el archivo *svm-train.c*, que hace uso de las funciones definidas en *svm.cpp*. Para reflejar de una manera visual la organización de los archivos que se van a analizar se han generado una serie de diagramas de dependencias con el generador de documentación Doxygen [61]. Para ello, fue necesario instalar la herramienta *graphviz* y crear un archivo de configuración adecuado. En la Figura 142 se muestra el esquema que relaciona los archivos *svm.cpp* y *svm-train.c* con el archivo de cabecera *svm.h*, en el que se encuentran las definiciones de las funciones y las estructuras necesarias. En la Figura 143 y en la Figura 144 se muestra el esquema de llamadas de funciones del archivo *svm.cpp* en el que se implica a la función *svm_train*.

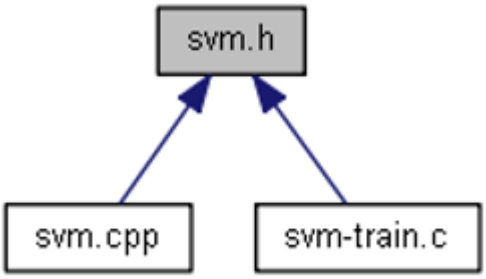

Figura 142. Fragmento del diagrama de dependencias de LIBSVM

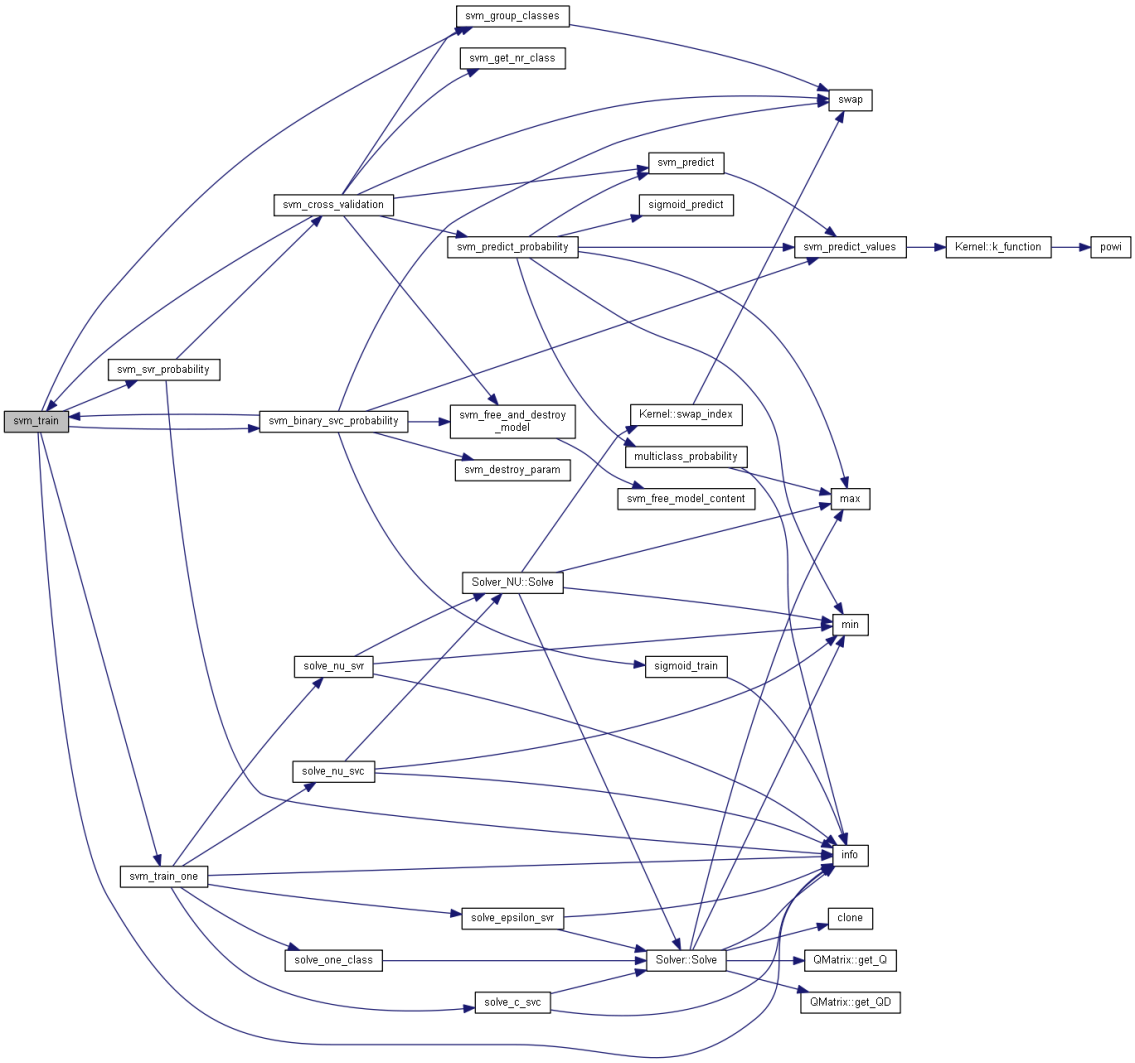

Figura 143. Fragmento del diagrama de dependencias de svm_train (I)

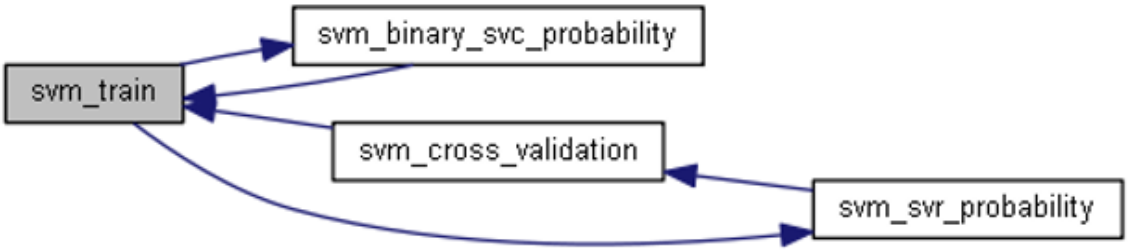

Figura 144. Fragmento del diagrama de dependencias de svm_train (II)

Una vez estructurado la organización del código, se pasó a analizar la manera en la que adaptarlo al dispositivo *Arduino*. Para ello, se identificaron las partes del código que resultasen necesarias para la clasificación utilizando el *kernel* y el método de clasificación seleccionados. De esta forma, para poder adaptar la función *svm_train* y evaluando el código original, se determinó que funciones como *solve_nu_svc, solve_one_class, solve_epsilon_svr, solve_nu_svr, svm_probability* podían eliminarse al no ser funciones útiles para la clasificación utilizando un *kernel* RBF y un método de clasificación C_SVC. Además, se eliminó la interfaz y se particularizó

la implementación para las opciones y parámetros *gamma* y C determinados, eliminándose el resto. Originalmente, estos parámetros se establecían en la función *parse_command_line*, y han pasado a establecerse directamente en la estructura *svm_parameter param*, que se muestra en la Figura 145, en la que se describen los parámetros utilizados para obtener el modelo.

```
struct svm_parameter param = {svm_type: C_SVC,
                              kernel_type: RBF,
                              degree: 3,
                              gamma: 0.0001,  // 1/num_features
                              coef0: 0,
                              nu: 0.5,
                              cache_size: 100,
                              C: 100,
                              eps: 1e-3,
                              p: 0.1,
                              shrinking: 1,
                              probability: 0,
                              nr_weight: 0,
                              weight_label: NULL,
                              weight: NULL
                              };
```

Figura 145. Código de la estructura svm_parameter param

De la misma manera pasó a definirse directamente la estructura *svm_problem prob*, tal y como se muestra en la Figura 146, cuyos parámetros eran anteriormente establecidos por la función *read_problem()*. En esta estructura, 'l' es el número de datos de entrenamiento, 'y' es una matriz que contiene los valores objetivo y 'x' es una matriz de punteros que apunta a una representación dispersa de un vector de entrenamiento dado.

```
struct svm_problem
{
  int l;
  double *y;
  struct svm_node **x;
};
```

Figura 146. Código de la estructura svm_problem

Tras incluir los valores de las muestras ya escaladas y correspondientes a la información a partir de la cual se entrena LIBSVM para un caso tomado como referencia, se generó el fichero *svm-train.c.* Con esta modificación, se representa la adaptación de la función *svm_train* de la biblioteca SVM original para su ejecución en el dispositivo *Arduino.*

Una vez realizado este paso, se llevó el código adaptado al dispositivo *Arduino* junto con las librerías *svm.cpp* y *svm.h*, que también han sido adaptadas para poder realizar la ejecución en el dispositivo de las funciones *svm_save_model(), svm_free_and_destroy_model()* y *svm_destroy_param()*.

Por otro lado, para poder adaptar el fichero *svm_scale.cpp*, y con el fin de poder realizar el escalado del valor de las muestras tomadas para el entrenamiento del clasificador de manera local en el dispositivo *Arduino*, se implementó la función *scaleValues().*

Se ha llevado a cabo también la adaptación de la función *svm_predict,* con el fin de generar los vectores de soporte sin necesidad de recurrir a un PC. Para ello, tomando como referencia los ficheros *svm.cpp* y *svm.h* de LIBSVM, se eliminaron todas las llamadas a las funciones relativas a la salida estándar (*stdout)*, como es el caso de *fflush* o *fputs*, además de eliminar funciones *kernel* como *k_function(), decisión_function()* o *svm_predict_values()*, para no tener en cuenta partes del código relativas a otros tipos de *kernel* diferentes de RBF.

Con las modificaciones de los ficheros *svm_scale* y *svm_predict* integradas en un *sketch .ino* del dispositivo *Arduino* se pudo realizar la validación del funcionamiento con el resto de módulos de la biblioteca LIBSVM adaptados con anterioridad. Para ello, fue necesario modificar el formato de los datos proporcionados a través de las variables *prob.l, prob.x* y *prob.y*, entre otros.

A partir de este *sketch* se realizaron diferentes pruebas iniciales en las que se llevó a cabo, tanto la adquisición de valores para la fase de entrenamiento, proceso de escalado y generación de los vectores de soporte, como el proceso de clasificación, íntegramente en el dispositivo *Arduino*. Finalmente, de esta forma se consiguió la caracterización de la funcionalidad del código correspondiente a la biblioteca LIBSVM, adaptado al dispositivo para un *kernel* de tipo RBF.

6.1.3 Funcionalidad de la plataforma final

En la Figura 147 se presenta el menú de usuario, en el que aparecen todas las funciones que se pueden llevar a cabo con la plataforma final. Las opciones principales del menú se encuentran indicadas con números y las secundarias con letras. Así, con la opción 0 se puede realizar la conexión BLE con el dispositivo *Myo*, mientras que con la opción a) se puede realizar el proceso de desconexión. Al seleccionar la opción 1 se podrán capturar muestras para generar nuevos vectores de entrenamiento. Antes de eso, tal y como se indica, es necesario ejecutar la opción c), con la que se eliminan los vectores anteriores, por si hubiese información ya existente almacenada en la memoria FRAM externa. Al almacenar ahora los vectoresen una memoria FRAM externa, se consigue una gran mejora en relación con la captura de los datos. Así, si se están capturando muestras y, por ejemplo, se produce una desconexión inesperada del dispositivo *Myo*, las muestras capturadas hasta ese momento no se perderán.

Con la opción b) del menú se pueden almacenar un conjunto de vectores de prueba definidos en el propio código, que podrán ser utilizados en los procesos de escalado, entrenamiento y posterior predicción. Se ha decidido mantener esta opción ya que facilita la comprobación del correcto funcionamiento de la plataforma sin necesidad de capturar nuevas muestras y crear un conjunto de vectores diferentes. La opción c), tal y como se comentó, permite borrar los vectores que se encuentren almacenados en la memoria FRAM. Esto también supone una gran ventaja, ya que, si por ejemplo se necesita volver a tomar muestras de una clase concreta, se podrá repetir únicamente el proceso de captura de esos datos, sin necesidad de crear todo el conjunto de vectores de nuevo. Si se elige la opción d) se mostrará por pantalla el número de vectores almacenados en la memoria FRAM externa para cada clase.

Al seleccionar la opción 2 del menú se imprimirán por pantalla los vectores de entrenamiento almacenados en la memoria FRAM para cada clase. Esta opción permitirá comprobar los datos de cada uno de los parámetros de los vectores, además de posibilitar la opción de visualizarlos y copiarlos para trasladarlos a la herramienta LIBSVM en el PC, tal y como se hacía en las anteriores implementaciones de la plataforma. La opción 3 habilita el escalado de los vectores, esta vez, directamente desde la plataforma. A partir del escalado, eligiendo la opción 4, se genera el modelo que se utilizará para la predicción de los gestos. Una vez generado, las opciones e) y f) permiten almacenar o cargar el modelo en la memoria FRAM externa, respectivamente. Por último, la opción 5 es la encargada de iniciar la predicción de los gestos. Una vez se tiene el modelo generado y cargado, se puede realizar el gesto correspondiente al símbolo deseado capturando las muestras, y comprobar en el *display* si el reconocimiento es correcto. Finalmente, se ha mantenido una opción en el menú identificada con la letra g), que permite realizar pruebas internas de detección a partir de los vectores de prueba definidos en el propio código.

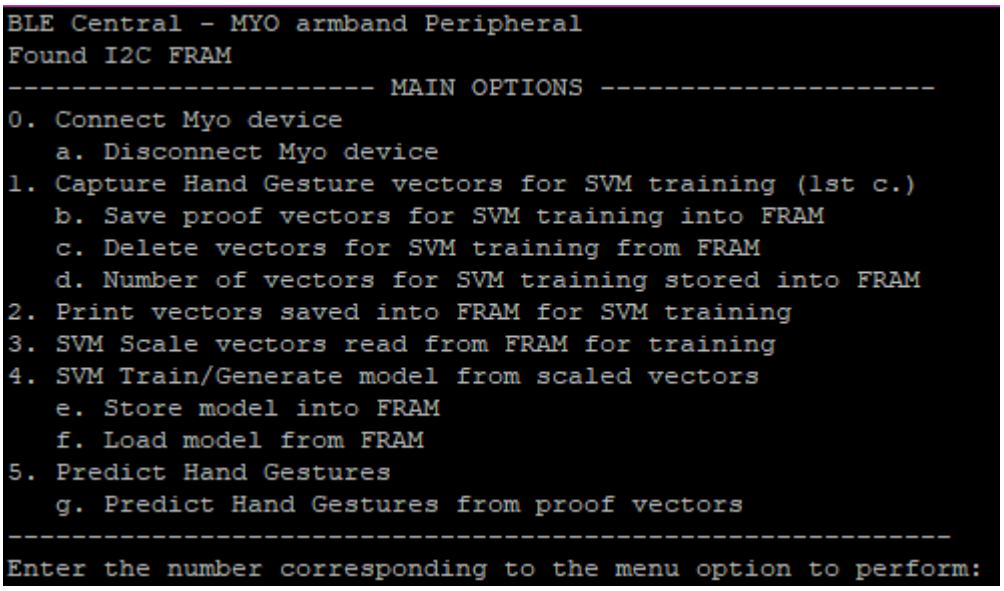

Figura 147. Menú para la plataforma final

A continuación, se pasa a describir la implementación de las nuevas funcionalidades que se han integrado en la plataforma *hardware/software* final. En primer lugar, se incluye en la Figura 148 un esquema general del flujo de llamadas del programa para tener una visión completa del funcionamiento de la plataforma. Hay que tener en cuenta que algunas funciones, como las relativas a la lectura y escritura de la memoria FRAM, no han sido incluidas para la simplificación del diagrama. Estas funciones pertenecen a la librería externa *Adafruit_FRAM_I2C*. Las dos funciones utilizadas principalmente en este caso son las de *read()* y *write()*. La primera lee múltiples bytes de la dirección de memoria especificada y devuelve el valor de 8 bits recuperado en la dirección. Para ello, recibe como parámetros la dirección de 16 bits desde la que leer, el puntero al *buffer* de bytes en el que se desea almacenar la información recibida, y el número de bytes a leer. La segunda escribe múltiples bytes en la dirección especificada y devuelve *true* o *false* en caso de éxito o fracaso de la operación, respectivamente. Para ello, recibe como parámetros la dirección de 16 bits en la que escribir en memoria, el puntero al *buffer* de bytes donde se encuentran almacenados los datos y el número de bytes a escribir. Cabe destacar también, que en la función *setup()* se ha incluido la llamada a la función *display_init()* que inicializa los parámetros relacionados con la pantalla de la plataforma. Las funciones asociadas a la gestión del *display* están disponibles gracias al uso de la librería *Adafruit_SSD1306*.

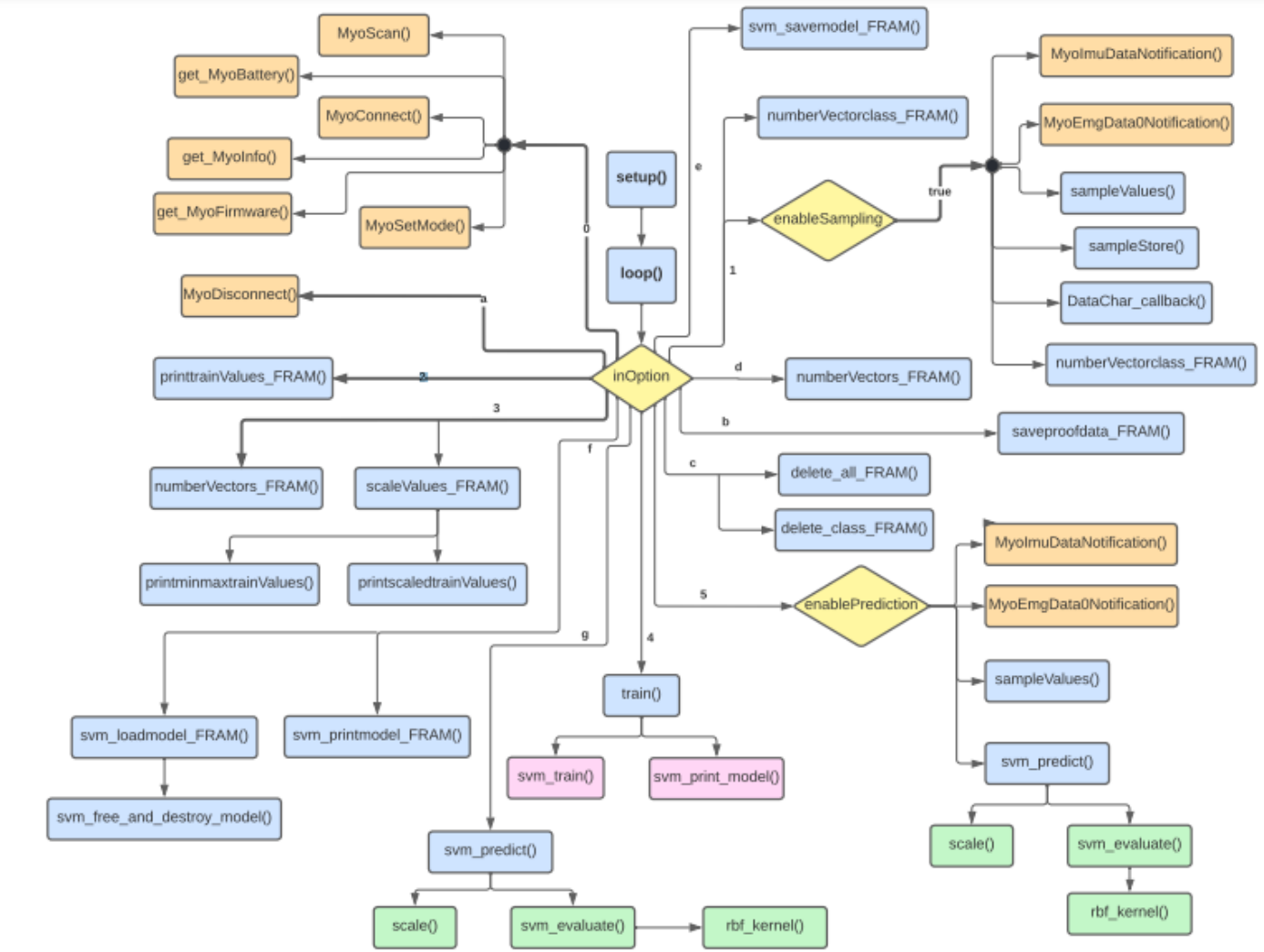

Figura 148. Diagrama de flujo de la plataforma final

El primer cambio realizado en el *sketch* de la plataforma se encuentra en la opción 1 del menú. Ahora se solicita directamente un número para la identificación de la clase asociada al gesto del que se desean capturar los datos, en lugar de una letra como anteriormente. En este caso, se ha definido una constante denominada *NUMBER_CLASS* con valor 5, que define el número máximo de clases para el proceso de clasificación. Por tanto, se solicitará al usuario introducir un identificador entre 0 y 4. Este identificador se envía como parámetro a la función *numberVectorsclass_FRAM()*, cuyo código se muestra en la Figura 149, para comprobar si ya existen datos almacenados en la memoria para la clase del gesto asociado con ese identificador. Esta función leerá el primer valor almacenado en memoria que se encuentre en la posición correspondiente al identificador del gesto deseado. Si el valor leído es un -1, se ignorarán los datos y el número de vectores almacenados para ese identificador será 0. En cambio, si es distinto de -1, el valor leído indicará el número de vectores disponibles para esa clase en la memoria externa. Ese valor, almacenado en la variable *n_vectors_class*, será el que la función *numberVectorsclass_FRAM()* retornará. De esta manera, se consigue realizar consultas eficientes en la memoria externa, sin necesidad de recorrer los vectores comprobando los datos que han sido almacenados.

```
int numberVectorsclass_FRAM (int n) {
   int n_vectors_class = 0;
   int rdata;
   int sizerdata = sizeof(rdata);
   uint16_t raddr = n*sizerdata;
   uint8_t buffer[sizerdata];
   fram.read(raddr, buffer, sizerdata);
   memcpy((void *)&rdata, buffer, sizerdata);
   if (rdata != -1) {
      n_vectors_class = rdata;
   }
   else {
      n_vectors_class = 0;
   }
   return(n_vectors_class);
}
```

Figura 149. Código de la función numberVectorsclass_FRAM()

En caso de que el número de valores devuelto por esta función no alcanza el máximo definido de vectores de entrenamiento (en este caso esta constante tiene un valor de 20 y se ha definido como *NUMBER_VECTORS*), y el valor introducido será un identificador válido, se habilita la toma de muestras asignando *true* a la variable booleana *enableSampling.* A partir de ahí, se repetirá el proceso ya descrito en las anteriores implementaciones para realizar el

proceso de captura de nuevos datos. En cambio, si el número de vectores almacenados para esa clase ya alcanza el máximo, se contará con el número de vectores suficientes para ese gesto y no se habilitará la toma de nuevos datos. En este caso, será necesario borrar los vectores almacenados previamente. Ambas comprobaciones se evalúan a partir de la sección de código que se muestra en la Figura 150.

```
else {
  int n = atoi(inID);
  n_values = numberVectorsclass_FRAM(n);
  if (n_values < NUMBER_VECTORS) {
     if ((bytesRead >= 1) && ((n >= 0) && (n < NUMBER_CLASS)) ) {
        enableSampling = true;
        available = 0;
        symbol = n; // the symbol associated to the sampled mean values
     }
     else {
        enableSampling = false;
        Serial.println("");
        Serial.println("Not a valid id");
        Serial.print("Enter the number corresponding to the menu option to perform: ");
     }
  }
  else {
     enableSampling = false;
     Serial.print("Vectors stored in FRAM are already available for class ");
     Serial.println(n);
     Serial.print("Enter the number corresponding to the menu option to perform: ");
  }
}
```

Figura 150. Fragmento de código que evalúa el retorno de numberVectorsclass_FRAM()

La nueva opción b) del menú permite almacenar los vectores de prueba en la memoria FRAM externa a partir de la función *saveproofdata_FRAM()*. Estos vectores de prueba se definen en el código, tal y como se muestra en la Figura 151, y han sido tomados de los archivos generados inicialmente, denominados *HandGesture_train_2_prob_y_arduino.txt* y *HandGesture_train_2_prob_x_arduino.txt*. De esta forma, se consigue tener siempre datos disponibles para realizar el resto de las funciones disponibles en el menú de usuario sin necesidad de tomar nuevas muestras.

```
int prob_y_proof[NUMBER_CLASS*NUMBER_VECTORS] PROGMEM =   {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                                           1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                                           2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                                                           3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
                                                           4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4};

int prob_x_proof[NUMBER_CLASS*NUMBER_VECTORS][NUMBER_EMG+NUMBER_ACC+NUMBER_MMACC] PROGMEM =
                                                          {{33, 11, 19, 13, 17, 5, 27, 42, 14, 13, 0, 1, 0},
                                                           {39, 15, 16, 12, 16, 6, 23, 51, 14, 14, 0, 0, 0},
                                                           {35, 12, 18, 13, 16, 6, 26, 52, 14, 14, 0, 1, 0},
                                                           {33, 10, 15, 9, 13, 7, 24, 51, 14, 15, 0, 0, 0},
                                                           {27, 8, 13, 8, 13, 7, 16, 39, 14, 15, 0, 0, 0},
                                                           {27, 11, 13, 9, 11, 5, 13, 36, 14, 15, 0, 0, 0},
                                                           {30, 12, 17, 12, 13, 8, 25, 37, 14, 16, 0, 0, 0},
```

Figura 151. Fragmento de código de los vectores de prueba

La opción c) permite borrar vectores almacenados en la memoria FRAM externa. Si se selecciona esta opción, se pedirá al usuario que introduzca el identificador de la clase del gesto del que se desean eliminar los datos, o que introduzca 'a' para eliminar todos los vectores almacenados en memoria. En el primer caso se hará uso de la función *delete_class_FRAM()*, cuyo código se muestra en la Figura 152. Esta recibe como parámetro el identificador de la clase elegida y se encargará de inicializar el primer elemento correspondiente a la posición del identificador del gesto elegido con el valor -1. De esta manera, dejará de tenerse en cuenta el número de vectores disponibles de esa clase en la memoria externa.

```
void delete_class_FRAM(int n) {
   i_data wdata;
   int sizewdata = sizeof(wdata.b);
   wdata.d = -1;
   uint16_t waddr = (uint16_t) n*sizewdata;
   fram.write(waddr, wdata.b, sizewdata);
}
```

Figura 152. Código de la función delete_class_FRAM()

De la misma manera, si se desea eliminar la totalidad de vectores se utilizará la función *delete_all_FRAM()*, cuyo código se muestra en la Figura 153. Esta función inicializará los primeros elementos almacenados en memoria para cada una de las clases con el valor -1. Así, cuando se consulte el número de vectores almacenados para alguna clase no se tendrán en cuenta los valores almacenados. Esta metodología, seguida tanto en *delete_class_FRAM()* como en *delete_all_FRAM()*, resulta también una opción eficiente en cuanto a cómputo, ya que no es necesario sobrescribir cada uno de los elementos de los vectores almacenados en memoria para realizar el borrado.

```
void delete_all_FRAM(void) {
   uint16_t waddr = 0;
   for (uint16_t i = 0; i < NUMBER_CLASS; i++){
      i_data wdata;
      wdata.d = -1;
      int sizewdata = sizeof(wdata.b);
      fram.write(waddr, wdata.b, sizewdata);
      waddr = waddr+sizewdata;
    }
    // Read deleted values to check
    int rdata;
    int sizerdata = sizeof(rdata);
    uint8_t buffer[sizerdata];
    uint16_t raddr = 0;
    for (uint16_t i = 0; i < NUMBER_CLASS; i++){
       fram.read(raddr, buffer, sizerdata);
       raddr = raddr+sizerdata;
        memcpy((void *)&rdata, buffer, sizerdata);
        Serial.print(rdata);
        Serial.print(" ");
    }
    Serial.println("");
}
```

Figura 153. Código de la función delete_all_FRAM()

Con la opción d) del menú se puede consultar mostrando por pantalla el número de vectores almacenados para cada una de las clases en la memoria FRAM externa. Para ello, se hace uso de la función *numberVectors_FRAM()*, que leerá de la memoria cada uno de los primeros elementos almacenados. El código de esta función se muestra en la Figura 154. De nuevo, este valor podrá ser -1 si no hay vectores almacenados para esa clase o contener el número de vectores disponibles. De esta manera, se irá incrementando la variable *n_vectors_FRAM* a modo de contador si el valor es distinto de -1, sumando el valor correspondiente al número de vectores para cada clase.

```
int numberVectors_FRAM(void) {
   int n_vectors_FRAM = 0;
   uint16_t raddr = 0;
   int rdata;
   int sizerdata = sizeof(rdata);
   uint8_t buffer[sizerdata];
   Serial.print("n_vectors = (");
   for (int i = 0; i < NUMBER_CLASS; i++)
   {
      fram.read(raddr, buffer, sizerdata);
      raddr = raddr+sizerdata;
      memcpy((void *)&rdata, buffer, sizerdata);
      Serial.print(rdata);
      Serial.print(" ");
      if (rdata != -1) {
         n_vectors_FRAM = n_vectors_FRAM + rdata;
      }
   }
   Serial.println(")");
   return(n_vectors_FRAM);
}
```

Figura 154. Código de la función numberVectors_FRAM()

Para imprimir el valor de estos vectores almacenados se debe seleccionar la opción 2 del menú. Haciendo la llamada a la función *printtrainValues_FRAM()*, cuyo código se muestra en la Figura 155, se mostrará el identificador de la clase y el contenido de los vectores que han sido leídos de la memoria.

```
for (int i = 0; i < NUMBER_CLASS; i++)
   {
   fram.read((uint16_t) i*sizerdata, buffer, sizerdata);
   memcpy((void *)&rdata, buffer, sizerdata);
   if (rdata != -1) {
      int n_vectors_class = rdata;
      raddr = NUMBER_CLASS*sizerdata + ((i * NUMBER_VECTORS *
              (1 + NUMBER_EMG + NUMBER_ACC + NUMBER_MMACC)) * sizerdata);
      for (int j = 0; j < n_vectors_class; j++)
         {
         fram.read(raddr, buffer, sizerdata);
         raddr = raddr+sizerdata;
         memcpy((void *)&rdata, buffer, sizerdata);
         Serial.print(rdata);
         Serial.print(" ");
         for (int k = 0; k < (NUMBER_EMG + NUMBER_ACC + NUMBER_MMACC); k++)
            {
            Serial.print(k+1);
            Serial.print(":");
            fram.read(raddr, buffer, sizerdata);
            raddr = raddr+sizerdata;
            memcpy((void *)&rdata, buffer, sizerdata);
            Serial.print(rdata);
            Serial.print(" ");
            }
         Serial.println();
         }
      }
```

Figura 155. Fragmento del código printtrainValues_FRAM()

Por otro lado, en la versión final se ha incorporado la funcionalidad del escalado desde la plataforma, habilitándose la opción 3 del menú de usuario. Así, en primer lugar, se llama a la función *numberVectors_FRAM()*, ya mencionada anteriormente, que devuelve el número de vectores de entrenamiento almacenados en memoria. En caso de que no haya suficientes vectores para todas las clases, se mostrará un mensaje para informar al usuario de este hecho y no se realizará el escalado. En cambio, si se cuenta con los vectores suficientes, se realiza la llamada a la función *scaleValues_FRAM()*. Con ella se calcula el valor máximo y mínimo de los datos de entrenamiento para establecer el rango de escalado y adecuar los valores de los vectores que se leen de memoria, tal y como se muestra en la Figura 156. Además, se realiza la llamada a la función *printminmaxtrainValues()*, que será la encargada de imprimir por pantalla el identificador de la posición del dato del vector, seguido de los valores máximos y mínimos encontrados para ese elemento concreto de los vectores de entrenamiento. De esta forma se podrá consultar el valor máximo y mínimo encontrado para todos los datos obtenidos relativos a cada uno de los sensores EMG, *pitch*, *roll*, y así sucesivamente.

```
// calculate max/min values from training data for each sensor
for (int j = 0; j < (NUMBER_EMG + NUMBER_ACC + NUMBER_MMACC); j++) {
   low_train[j] = INT_MAX;
   high_train[j] = -INT_MAX;
}

for (int i = 0; i < n_vectors_FRAM; i++) {
   raddr = raddr+4; // for avoiding stored id for each vector
   for (int j = 0; j < (NUMBER_EMG + NUMBER_ACC + NUMBER_MMACC); j++) {
      fram.read(raddr, buffer, sizerdata);
      raddr = raddr+4;
      memcpy((void *)&rdata, buffer, sizerdata);
      if (rdata < low_train[j])
        low_train[j] = rdata;
      if (rdata > high_train[j])
        high_train[j] = rdata;
   }
}
printminmaxtrainValues();
```

Figura 156. Fragmento del código de la función scaleValues_FRAM() (I)

Una vez calculados los valores máximos y mínimos, se procede a realizar el proceso del escalado de todos los datos de entrenamiento, tal y como se muestra en la Figura 157, realizando una comparación de cada uno de los elementos que conforman los vectores con los valores que establecen el rango de escalado. La variable *scaledprob_y[]* almacenará el identificador del gesto correspondiente y la variable *scaledprob_x[][]* almacenará los diferentes valores escalados de cada uno de los vectores.

```
raddr = OFFSET_N_CLASS;
for (int i = 0; i < n_vectors_FRAM; i++) {
   fram.read(raddr, buffer, sizerdata);
   raddr = raddr+4;
   memcpy((void *)&rdata, buffer, sizerdata);

   scaledprob_y[i] = (float) rdata;

   for (int j = 0; j < (NUMBER_EMG + NUMBER_ACC + NUMBER_MMACC); j++) {
      fram.read(raddr, buffer, sizerdata);
      raddr = raddr+4;
      memcpy((void *)&rdata, buffer, sizerdata);

      if (rdata == low_train[j]) {
        scaledprob_x[i][j] = (float) scaleParprob_x[0];
      }
      else if (rdata == high_train[j]) {
        scaledprob_x[i][j] = (float) scaleParprob_x[1];
      }
      else {
        scaledprob_x[i][j] = ((float)scaleParprob_x[0] +((float)scaleParprob_x[1] -((float)scaleParprob_x[0]))*
                             ((float)rdata-(float)low_train[j])/((float)high_train[j]-(float)low_train[j]));
      }
   }
}
printscaledtrainValues(scaledprob_y, scaledprob_x, n_vectors_FRAM);
```

Figura 157. Fragmento del código de la función scaleValues_FRAM() (II)

Por último, se realiza una llamada a la función *printscaledtrainValues()* para mostrar por pantalla los nuevos vectores con valores escalados. El código de esta función, similar al de *printminmaxtrainValues()*, se muestra en la Figura 158. Se mostrará el identificador del gesto correspondiente seguido de los nuevos valores para cada uno de los trece elementos de los vectores de entrenamiento.

```
void printscaledtrainValues(float *scaledprob_y, float scaledprob_x[][NUMBER_EMG + NUMBER_ACC + NUMBER_MMACC],
                            int n_vectors) {
    for (int i = 0; i < n_vectors; i++) {
      Serial.print((int) scaledprob_y[i]);
      Serial.print(" ");
      for (int j = 0; j < (NUMBER_EMG + NUMBER_ACC + NUMBER_MMACC); j++) {
         Serial.print(j+1);
         Serial.print(":");
         Serial.print(scaledprob_x[i][j], 6);
         Serial.print(" ");
      }
      Serial.println();
    }
}
```

Figura 158. Código de la función printscaledtrainValues()

A partir de los vectores escalados se puede realizar el proceso de entrenamiento y la generación del modelo. Esta funcionalidad también se ha incorporado como novedad en la plataforma *hardware/software* final en la opción 4 del menú de usuario. En este caso se realiza la llamada a la función *train(),* pasándole como parámetro los vectores de entrenamiento escalados. Con esta función se calculan los dos parámetros de entrenamiento necesarios para generar el modelo, identificados mediante la variable *prob* y *param*. El primer parámetro será una estructura que contendrá el número de datos de entrenamiento, una matriz que contiene sus valores objetivo, o identificadores de clase, y una matriz de punteros que apuntan a cada uno de los vectores de entrenamiento. La configuración de los elementos de esta estructura en la función *train()* se muestran en la Figura 159.

```
void train(float *scaledprob_y, float scaledprob_x[][NUMBER_EMG + NUMBER_ACC + NUMBER_MMACC]) {
   struct svm_problem prob;
   struct svm_node *x_space;
   double tmp [NUMBER_CLASS*NUMBER_VECTORS];
   for (int i = 0 ; i < NUMBER_CLASS*NUMBER_VECTORS; i++) {
      tmp[i] = (double) scaledprob_y[i];
   }

   prob.l = NUMBER_CLASS*NUMBER_VECTORS;
   prob.y = tmp;
   prob.x = Malloc(struct svm_node *, NUMBER_CLASS*NUMBER_VECTORS);

   for (int i = 0; i < NUMBER_CLASS*NUMBER_VECTORS; i++) {
      x_space = Malloc(struct svm_node, VEC_DIM + 1);
      for (int j = 0; j < VEC_DIM; j++) {
        x_space[j].index = j+1;
        x_space[j].value = (double) scaledprob_x[i][j];
      }
      x_space[VEC_DIM].index = -1;
      x_space[VEC_DIM].value = 0;

      prob.x[i] = x_space;
   }
```

Figura 159. Fragmento del código de la función train()

La segunda variable determina los parámetros del modelo SVM. En este caso se ha establecido una clasificación de tipo C-SVC y el conjunto de factores necesarios para la generación del modelo que se muestran en la Figura 160.

```
struct svm_parameter param = {svm_type: C_SVC,
                              kernel_type: RBF,
                              degree: 3,
                              gamma: 0.0001,  // 1/num_features
                              coef0: 0,
                              nu: 0.5,
                              cache_size: 100,
                              C: 100,
                              eps: 1e-3,
                              p: 0.1,
                              shrinking: 1,
                              probability: 0,
                              nr_weight: 0,
                              weight_label: NULL,
                              weight: NULL
                              };
```

Figura 160. Establecimiento de parámetros para la generación del modelo SVM

Ambos parámetros se envían a la función *svm_train()* de la librería SVM adaptada a la plataforma, que será la encargada de construir y devolver un modelo SVM de acuerdo a los datos y parámetros de entrenamiento dados. En la Figura 161 se muestra un fragmento del código de esta función, en la que se utilizan los *nr_class**(*nr_class*-1)/2 clasificadores para realizar el proceso de entrenamiento de los datos, siguiendo la estrategia de votación.

```
// train k*(k-1)/2 models

bool *nonzero = Malloc(bool,l);
for(i=0;i<l;i++)
  nonzero[i] = false;
decision_function *f = Malloc(decision_function,nr_class*(nr_class-1)/2);

double *probA=NULL,*probB=NULL;
if (param->probability)
{
  probA=Malloc(double,nr_class*(nr_class-1)/2);
  probB=Malloc(double,nr_class*(nr_class-1)/2);
}

int p = 0;
for(i=0;i<nr_class;i++)
  for(int j=i+1;j<nr_class;j++)
  {
    svm_problem sub_prob;
    int si = start[i], sj = start[j];
    int ci = count[i], cj = count[j];
    sub_prob.l = ci+cj;
    sub_prob.x = Malloc(svm_node *,sub_prob.l);
    sub_prob.y = Malloc(double,sub_prob.l);
    int k;
    for(k=0;k<ci;k++)
    {
      sub_prob.x[k] = x[si+k];
      sub_prob.y[k] = +1;
```

Figura 161. Fragmento del código svm_train()

Una vez obtenido el modelo se envía a la función *svm_print_model()*, que imprimirá por pantalla los diferentes parámetros del modelo (número de clases, coeficientes, constantes en funciones de decisión, etc.), tal y como se muestra en el fragmento de código de la Figura 162.

```
if(param.kernel_type == POLY || param.kernel_type == RBF || param.kernel_type == SIGMOID)
  {
    Serial.print("gamma ");
    Serial.println(param.gamma);
  }
if(param.kernel_type == POLY || param.kernel_type == SIGMOID)
  {
    Serial.print("coef0 ");
    Serial.println(param.coef0);
  }

int nr_class = model->nr_class;
int l = model->l;
Serial.print("nr_class ");
Serial.println(nr_class);
Serial.print("total_sv ");
Serial.println(l);

{
  Serial.print("rho");
  for(int i=0;i<nr_class*(nr_class-1)/2;i++)
    {
      Serial.print(" ");
      Serial.print(model->rho[i]);
    }
  Serial.println("");
}
```

Figura 162. Fragmento de código de la función svm_print_model()

Con el modelo generado, las opciones e) y f) del menú de usuario permitirán cargar y almacenar el modelo en la memoria FRAM externa, respectivamente. Al seleccionar la opción e) se llamará a la función *svm_savemodel_FRAM()*, que almacenará los parámetros del modelo realizando las escrituras correspondientes en memoria. En la Figura 163 se muestra un fragmento del código de la función donde se almacenan en la memoria FRAM externa parámetros como *nr_class, rho* o *total_sv.*

```
wdata_i.d = model->nr_class; // nr_class
fram.write(waddr, wdata_i.b, sizeof(wdata_i.b));
waddr = waddr+sizeof(wdata_i.b);

wdata_i.d = model->l; // total_sv
fram.write(waddr, wdata_i.b, sizeof(wdata_i.b));
waddr = waddr+sizeof(wdata_i.b);

for(int i=0;i<model->nr_class*(model->nr_class-1)/2;i++) {
  wdata_d.d = model->rho[i]; // rho[i]
  fram.write(waddr, wdata_d.b, sizeof(wdata_d.b));
  waddr = waddr+sizeof(wdata_d.b);
  }

for(int i=0;i<model->nr_class;i++) {
  wdata_i.d = model->label[i]; // label[i]
  fram.write(waddr, wdata_i.b, sizeof(wdata_i.b));
  waddr = waddr+sizeof(wdata_i.b);
  }

for(int i=0;i<model->nr_class;i++) {
  wdata_i.d = model->nSV[i]; // nSV[i]
  fram.write(waddr, wdata_i.b, sizeof(wdata_i.b));
  waddr = waddr+sizeof(wdata_i.b);
  }
```

Figura 163. Fragmento de código de la función svm_savemodel_FRAM()

Seleccionando la opción f) se llamará a la función *svm_loadmodel_FRAM()* con el fin de obtener el modelo almacenado y se mostrará por pantalla mediante la función *svm_printmodel_FRAM()*. De la misma manera que anteriormente con *svm_savemodel_FRAM()*, utilizando ahora la función *read()*, se leerá de la memoria externa los diferentes parámetros almacenados, tal y como se muestra en la Figura 164.

```
fram.read(raddr, buffer_i, sizeof(rdata_i));
memcpy((void *)&rdata_i, buffer_i, sizeof(rdata_i));
model->nr_class = rdata_i; // nr_class
raddr = raddr+sizeof(rdata_i);

fram.read(raddr, buffer_i, sizeof(rdata_i));
memcpy((void *)&rdata_i, buffer_i, sizeof(rdata_i));
model->l = rdata_i; // total_sv
raddr = raddr+sizeof(rdata_i);

model->rho = Malloc(double, model->nr_class*(model->nr_class-1)/2);
for(int i=0;i<model->nr_class*(model->nr_class-1)/2;i++) {
  fram.read(raddr, buffer_d, sizeof(rdata_d));
  memcpy((void *)&rdata_d, buffer_d, sizeof(rdata_d));
  model->rho[i] = rdata_d; // rho[i]
  raddr = raddr+sizeof(rdata_d);
  }

model->label = Malloc(int, model->nr_class);
for(int i=0;i<model->nr_class;i++) {
  fram.read(raddr, buffer_i, sizeof(rdata_i));
  memcpy((void *)&rdata_i, buffer_i, sizeof(rdata_i));
  model->label[i] = rdata_i; // label[i]
  raddr = raddr+sizeof(rdata_i);
  }
```

Figura 164. Fragmento de código de la función svm_loadmodel_FRAM()

La función *svm_loadmodel_FRAM()* hace uso, a su vez, de *svm_free_and_destroy_model()* de la librería SVM. Esta función libera la memoria utilizada por un modelo y destruye su estructura, tal y como se muestra en la Figura 165.

```
void svm_free_and_destroy_model(svm_model** model_ptr_ptr)
{
  if(model_ptr_ptr != NULL && *model_ptr_ptr != NULL)
  {
    svm_free_model_content(*model_ptr_ptr);
    free(*model_ptr_ptr);
    *model_ptr_ptr = NULL;
  }
}
```

Figura 165. Código de la función svm_free_and_destroy_model()

La opción 5 del menú de usuario, correspondiente a la predicción de los gestos, se había implementado anteriormente. Sin embargo, ahora, en la función *svm_predict()* se leerá el modelo previamente generado y almacenado en la memoria FRAM externa y se realizará el proceso de votación a partir de sus parámetros. En este caso, se incorpora el uso del *display* que permitirá la visualización de la clase asociada al gesto reconocido. Para garantizar el correcto funcionamiento de la predicción de gestos, se ha mantenido en el menú de usuario la opción g), que sigue el mismo procedimiento de detección ya detallado, pero utilizando directamente unos

vectores de entrenamiento ya definidos. Estos vectores de prueba se muestran en la Figura 166 junto con sus identificadores, y se corresponden con nuevos gestos hasta ahora no definidos.

```
int sensors_proof[2*5][NUMBER_EMG+NUMBER_ACC+NUMBER_MMACC] = {{39, 15, 16, 12, 16, 6, 23, 51, 14, 14, 0, 0, 0}, //0
                                                              {27, 8, 13, 8, 13, 7, 16, 39, 14, 15, 0, 0, 0}, //0
                                                              {10, 8, 6, 11, 19, 6, 36, 25, 9, 10, 0, 0, 0}, //1
                                                              {9, 8, 5, 10, 20, 8, 41, 31, 10, 10, 0, 0, 0}, //1
                                                              {44, 11, 10, 14, 28, 28, 26, 24, 5, 46, 0, 0, 0}, //2
                                                              {29, 9, 13, 11, 21, 28, 16, 21, 4, 47, 0, 0, 0}, //2
                                                              {9, 8, 10, 46, 32, 8, 21, 25, 6, 21, 0, 0, 0}, //3
                                                              {11, 10, 10, 48, 37, 8, 24, 23, 6, 21, 0, 0, 0}, //3
                                                              {5, 5, 5, 17, 13, 4, 4, 9, 13, 14, 0, 0, 0}, //4
                                                              {8, 5, 6, 16, 14, 2, 5, 8, 13, 15, 0, 0, 0}}; //4
```

Figura 166. Vectores de prueba para realizar las predicciones

La definición de los nuevos gestos, así como el número correspondiente al identificador que se les ha asignado, se muestra en la Tabla 10. Estos gestos se corresponden con cinco posiciones diferentes de la palma de la mano, colocándola hacia arriba, abajo, derecha, izquierda o de frente. Se han definido estas posiciones al ser gestos sencillos de realizar, estáticos y diferentes a los hasta ahora utilizados.

Tabla 10. Nuevos gestos de prueba

Identificador	Descripción
0	*Palm up*
1	*Palm down*
2	*Palm left*
3	*Palm right*
4	*Palm front*

6.1.4 Validación de la plataforma final

Para llevar a cabo la validación de la plataforma final se ha planteado este apartado de la memoria como una referencia de usuario para utilizar la totalidad de las funcionalidades de la plataforma *hardware/software*. Por tanto, se detallarán los pasos seguidos para comprobar todas y cada una de las opciones que se plantean en el menú final mostrado anteriormente, presentando algunas de las salidas que se obtienen a través del terminal serie para validar las diferentes funcionalidades de la plataforma *hardware/software* final desarrollada en este TFG.

Una vez cargado el programa en el dispositivo *Arduino* y establecida la conexión con el terminal serie, el primer paso será establecer la conexión BLE con el dispositivo *Myo Armband*. Este proceso se detalló con anterioridad, al validar el funcionamiento de la plataforma implementada inicialmente, mostrando la información referente a la batería restante o el

firmware, entre otros, obtenidos desde el periférico. De la misma forma, si se desea realizar en cualquier momento la desconexión del dispositivo *Myo Armband* se tendrá que seleccionar la opción a) del menú de usuario.

Una vez realizada correctamente la conexión BLE entre el brazalete *Myo* y el dispositivo *Arduino*, se deberá seleccionar la opción 1 del menú para realizar la toma de nuevos datos con los que se definirán los vectores de entrenamiento de los gestos que se vayan a definir para la clasificación. Sin embargo, será necesario seleccionar previamente la opción c) del menú, para asegurar que no hay ya otros vectores almacenados en la memoria FRAM externa. Para ello, se solicitará que se introduzca el identificador de la clase del gesto del que se desean borrar los vectores de entrenamiento, o bien que se introduzca la letra 'a' para eliminar todos los datos de cada una de las clases, tal y como se muestra en la Figura 167.

```
Enter the number corresponding to the menu option to perform: c
Enter the id of the hand gesture data to delete(0 to 4, or 'a' to delete all): a
-1 -1 -1 -1 -1

Enter the number corresponding to the menu option to perform: 1
Enter the id for the hand gesture data (0 to 4): 0
*** n_vectors for class 0 = 0
```

Figura 167. Prueba opción c del menú

Una vez asegurado esto, se procederá a capturar los datos correspondientes a cada uno de los gestos que realice el usuario. Al seleccionar la opción número 1 se solicitará al usuario que introduzca un identificador para la clase asociada al gesto, que en este caso deberá estar comprendido entre el 0 y el 4. Además, se mostrará por pantalla el número de vectores que ya han sido almacenados en la memoria para el identificador de esa clase, por si el usuario se encuentra en mitad del proceso de captura de muestras o se ha realizado la desconexión con el dispositivo *Myo*, garantizando que no existen pérdidas de datos. Posteriormente, el programa estará a la espera de que se presione el pulsador y se comenzarán a capturar los datos correspondientes a las características suscritas en el dispositivo *Myo*, referentes a los sensores EMG y a la IMU. En la Figura 168 se muestra de forma gráfica el proceso descrito, en el que un usuario realiza a modo de ejemplo un gesto con su mano y se capturan con la plataforma los datos correspondientes durante su ejecución.

Figura 168. Proceso de captura de muestras para un gesto determinado

Por otra parte, si no se desea capturar nuevas muestras relativas a unos gestos determinados para el proceso de entrenamiento, se pueden emplear los vectores de prueba que están definidos por defecto y que hacen referencia a los gestos que se detallaron en la Tabla 10. En la Figura 169 se puede comprobar cómo se almacenan en la memoria los vectores de prueba directamente al seleccionar la opción b) del menú de usuario y, tras ello, cómo se indica el número de elementos almacenados al seleccionar la opción d). Tal y como se había definido en el código, se tendrán 20 vectores correspondientes a cada una de las 5 clases posibles. La opción d) del menú podrá ser utilizada también en cualquier momento si se están realizando nuevas capturas de datos con el brazalete *Myo.*

```
Enter the number corresponding to the menu option to perform: b
Proof Data for SVM training stored into FRAM (5616)

Enter the number corresponding to the menu option to perform: d
n_vectors = (20 20 20 20 20 )
Number of train vectors stored into FRAM = 100
```

Figura 169. Prueba opción b y d del menú.

Independientemente de cómo se hayan obtenido los datos para el proceso de entrenamiento, ya sea mediante la captura de muestras o mediante la carga de los vectores de entrenamiento predefinidos, el siguiente paso se centra en el escalado de los datos. Cuando se

realiza el escalado de los vectores que se hayan almacenado en la memoria FRAM externa seleccionando la opción 3 del menú, se mostrará por pantalla el resultado de los diferentes parámetros escalados en función del rango calculado entre los valores máximos y mínimos de las clases. Un fragmento de la salida por pantalla se representa en la Figura 170.

```
Enter the number corresponding to the menu option to perform: 3
n_vectors = (20 20 20 20 20 )
0 10
1 4 44
2 4 15
3 3 28
4 5 48
5 7 50
6 2 30
7 2 42
8 2 52
9 -2 15
10 9 47
11 0 1
12 0 2
13 0 1

0 1:7.250000 2:6.363636 3:6.400000 4:1.860465 5:2.325581 6:1.071429 7:6.250000 8
:8.000000 9:9.411765 10:1.052632 11:0.000000 12:5.000000 13:0.000000
0 1:8.750000 2:10.000000 3:5.200000 4:1.627907 5:2.093023 6:1.428571 7:5.250000
8:9.800000 9:9.411765 10:1.315789 11:0.000000 12:0.000000 13:0.000000
0 1:7.750000 2:7.272727 3:6.000000 4:1.860465 5:2.093023 6:1.428571 7:6.000000 8
:10.000000 9:9.411765 10:1.315789 11:0.000000 12:5.000000 13:0.000000
0 1:7.250000 2:5.454545 3:4.800000 4:0.930233 5:1.395349 6:1.785714 7:5.500000 8
:9.800000 9:9.411765 10:1.578947 11:0.000000 12:0.000000 13:0.000000
0 1:5.750000 2:3.636364 3:4.000000 4:0.697674 5:1.395349 6:1.785714 7:3.500000 8
:7.400000 9:9.411765 10:1.578947 11:0.000000 12:0.000000 13:0.000000
0 1:5.750000 2:6.363636 3:4.000000 4:0.930233 5:0.930233 6:1.071429 7:2.750000 8
:6.800000 9:9.411765 10:1.578947 11:0.000000 12:0.000000 13:0.000000
```

Figura 170. Prueba opción 3 del menú

Una vez se tienen los datos escalados, se continúa con el proceso de entrenamiento mediante la selección de la opción 4 del menú de inicio. En este caso, se muestran por pantalla los diferentes parámetros correspondientes a la generación del modelo. Algunos de ellos se pueden visualizar en el fragmento de la salida obtenida por el terminal que se muestra en la Figura 171.

```
Enter the number corresponding to the menu option to perform: 4
svm_type c_svc
kernel_type rbf
gamma 0.00
nr_class 5
total_sv 35
rho -0.30 -0.06 0.02 0.02 0.00 -0.49 -0.39 -0.15 0.04 -0.02
label 0 1 2 3 4
nr_sv 6 4 5 9 11
SV
sv_coeff1 = {39.35, 64.98, 0.00, 0.00, 0.00, 77.20, 14.34, 0.00, 15.18, 27.74, 8
0.00, 0.00, 37.09, 0.00, 0.00, 0.00, 33.73, 100.00, 74.36, 0.00, 0.00, 0.00, 29.
61, 100.00, }
sv1 = {1:4.25, 2:2.73, 3:4.00, 4:1.16, 5:1.40, 6:1.79, 7:3.25, 8:8.00, 9:9.41, 1
0:1.84, 11:0.00, 12:0.00, 13:0.00,
       1:6.00, 2:5.45, 3:2.80, 4:1.16, 5:1.63, 6:2.86, 7:4.50, 8:8.80, 9:10.00,
10:1.84, 11:0.00, 12:0.00, 13:0.00,
       1:6.50, 2:4.55, 3:8.40, 4:1.63, 5:2.56, 6:1.79, 7:2.75, 8:3.80, 9:8.82, 1
0:1.84, 11:0.00, 12:0.00, 13:0.00,
       1:6.25, 2:4.55, 3:6.40, 4:2.33, 5:3.49, 6:2.14, 7:2.25, 8:3.80, 9:8.82, 1
0:1.84, 11:0.00, 12:0.00, 13:0.00,
       1:4.75, 2:2.73, 3:6.80, 4:1.63, 5:3.26, 6:2.14, 7:2.00, 8:3.20, 9:9.41, 1
0:1.84, 11:0.00, 12:0.00, 13:0.00,
       1:4.00, 2:2.73, 3:6.00, 4:2.09, 5:3.02, 6:1.79, 7:1.75, 8:3.00, 9:10.00,
10:1.84, 11:0.00, 12:0.00, 13:0.00,
       }
```

Figura 171. Prueba opción 4 del menú

Una vez generado el modelo, este puede almacenarse en la memoria FRAM externa seleccionando la opción e), para posteriormente ser utilizado en las predicciones, como se muestra en la Figura 172.

```
Enter the number corresponding to the menu option to perform: e
Model stored into FRAM (4992)
```

Figura 172. Prueba opción e del menú

Con el modelo generado y almacenado en la memoria FRAM externa, el último paso consistirá en realizar las predicciones de los gestos, eligiendo la opción 5 del menú de usuario. Al seleccionar esta opción se deberá presionar el pulsador dispuesto en la plataforma para que se puedan tomar los datos del gesto de la mano que se va a realizar a partir del dispositivo *Myo Armband* y, así, poder llevar a cabo el proceso de clasificación en base a las clases que han sido utilizadas en el proceso de entrenamiento. Un fragmento de la salida obtenida en el terminal se muestra en la Figura 173, donde finalmente aparece el número de la clase que se identifica con el gesto realizado. En este caso, se ha detectado la posición de la mano denominada *palm left*, tal y como se puede comprobar en la pantalla del *display* de la Figura 174.

```
19, 24, 60, 8, 17, 8, 12, 1, -10.93, 50.50, 9.20, 36.04, 28.22
6, 18, 16, 10, 42, 17, 47, 90, -10.94, 50.45, 8.89, 35.30, 28.20
76, 12, 1, 13, 35, 10, 34, 9, -10.94, 50.48, 8.72, 35.67, 28.08
1:36 2:18 3:12 4:10 5:24 6:15 7:39 8:36 9:-11 10:49 11:1 12:1 13:0
· Unsubscribing to IMUDataChar characteristic for Myo 0 ...
· Unsubscribing to EMGData0Char characteristic for Myo 0 ...
36, 18, 12, 10, 24, 15, 39, 36, -11, 49, 1, 1, 0,

1:8.000000 2:12.727273 3:3.600000 4:1.162791 5:3.953488 6:4.642857 7:9
> recognized class = 2
```

Figura 173. Prueba predicción con la opción 5 del menú

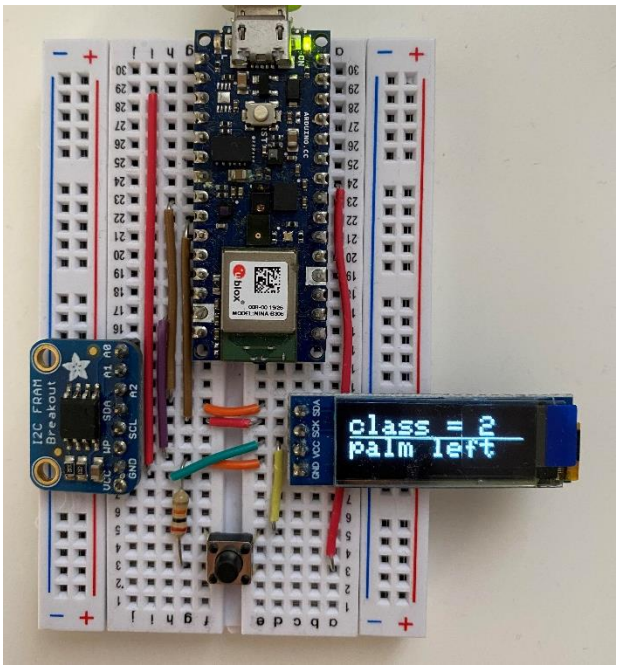

Figura 174. Reconocimiento del gesto en el 137isplay

Este proceso se comprobó con el resto de los símbolos de prueba, tomando también muestras de los gestos estáticos y dinámicos definidos anteriormente para realizar nuevas detecciones. De esta manera, se ha conseguido completar un proceso de validación de la plataforma *hardware/software* final que se ha desarrollado en el presente Trabajo Fin de Grado con el que comprobar que se cumplen los objetivos planteados inicialmente. En definitiva, se ha implementado una plataforma completa de bajo coste, escalable, y capaz de generar diferentes modelos para llevar a cabo predicciones de gestos.

CAPÍTULO 7 Conclusiones

En este capítulo se presentan las conclusiones obtenidas tras el desarrollo y el alcance de los objetivos propuestos para el presente Trabajo Fin de Grado. Además, se plantean y definen una serie de aspectos a modo de líneas futuras de trabajo que suponen mejoras para la implementación de la plataforma implementada.

7.1 Conclusiones

Al finalizar el desarrollo de la plataforma *hardware/software* para el reconocimiento de gestos basado en electromiografía y tras haber realizado una serie de pruebas experimentales para su validación funcional, puede concluirse que con la realización de este Trabajo Fin de Grado se han alcanzado los objetivos definidos y establecidos inicialmente.

Se ha implementado una plataforma de bajo coste capaz de reconocer diferentes tipos de gestos utilizando el dispositivo *Myo Armband* gracias a sus sensores EMG y su IMU, integrando el proceso de entrenamiento y clasificación SVM en un único dispositivo IoT basado en MCU, consiguiendo un dispositivo totalmente personalizable.

En un primer momento, se desarrolló una plataforma inicial a modo de primera aproximación en la que se obtenían las muestras necesarias gracias al brazalete *Myo* y se llevaba a cabo el proceso de entrenamiento, generación del modelo y clasificación de los gestos utilizando herramientas externas que facilitaban el proceso de detección de los símbolos.

Posteriormente, se realizaron las adaptaciones de *firmware* correspondientes para conseguir integrar los procesos que se realizaban externamente directamente en la plataforma final. De esta forma, se consigue implementar en la plataforma la conexión y desconexión con el dispositivo *Myo Armband*, la captura de muestras para generar los vectores de entrenamiento, la lectura o el almacenamiento de los vectores en memoria, el escalado de los vectores para el entrenamiento, el entrenamiento SVM y la generación del modelo, la lectura o el almacenamiento del modelo generado en memoria y la predicción de los gestos. Además, para poder conseguir los objetivos planteados, se realizaron modificaciones en cuanto al *hardware* que aportasen las prestaciones necesarias para alcanzar la integración de todas las funcionalidades requeridas en la plataforma. Para ello, entre otros aspectos, se incorporó una memoria externa, que permitiese almacenar vectores de entrenamiento y modelos generados, un *display,* que aportase una mejor interacción con el usuario.

Finalmente, se realizaron diversas pruebas experimentales para validar el funcionamiento de la plataforma *hardware/software* final y puede concluirse también que, con un número máximo de 5 clases, obteniendo 50 muestras en cada uno de los 20 vectores para la clasificación, pueden realizarse predicciones exitosas.

Por tanto, se ha conseguido una implementación que integra toda la funcionalidad requerida para llevar a cabo un proceso de clasificación, sin necesidad de un PC u otras herramientas externas, lo que supone una gran ventaja en cuanto a eficiencia y escalabilidad, teniendo una plataforma *hardware/software* totalmente personalizable. Este aspecto distingue, principalmente, el trabajo desarrollado en este TFG frente a otras propuestas.

Sin embargo, durante el proceso de desarrollo también fueron surgiendo diferentes problemas que han tenido que abordarse. Por ejemplo, con la obtención de los datos que se extraen mediante el dispositivo *Myo Armband*, fue necesario tener en cuenta el muestreo de los datos a una misma frecuencia y actualizar a la vez las características relativas al sensor EMG y a la IMU para conseguir muestras correctas y relativas a un mismo momento. También, inicialmente, se cometieron algunos errores en el proceso de escalado al utilizar los comandos LIBSVM, generando un *range* incorrecto que ralentizó algunos de los procesos de validación de la plataforma.

En todo caso, el mayor problema encontrado es el relativo a la memoria, habiendo tenido que modificar el dispositivo *Arduino* empleado e incorporar el uso de una memoria externa para poder almacenar y gestionar toda la información de los vectores de entrenamiento y modelos generados de la forma más eficiente posible para no provocar pérdidas de datos. Finalmente, ha sido necesario tener en cuenta ciertas modificaciones en relación al formato o al tipo de los datos al realizar las adaptaciones de código relativas a LIBSVM.

7.2 Líneas futuras

Tras alcanzar los objetivos planteados, se da por finalizado el presente Trabajo Fin de Grado, proponiéndose algunas posibles líneas futuras.

- Implementar una integración de la plataforma *hardware/software* con el asistente virtual Alexa de Amazon. Este asistente es uno de los asistentes virtuales más populares y destaca por ser un servicio de voz ubicado en la nube de Amazon. Permite crear experiencias de voz naturales para ofrecer a los clientes una forma más intuitiva de interactuar con la tecnología diaria. Integrando la plataforma desarrollada junto con el asistente virtual se conseguiría que este fuese capaz de

recibir información de forma diferente y no únicamente en forma de voz, mejorando la herramienta y permitiendo que fuese aún más adaptable a cualquier entorno de trabajo, lugar o aplicación.

- Ampliar el número de gestos o símbolos que puedan ser detectados por la plataforma desarrollada de forma óptima, teniendo en cuenta los requisitos de eficiencia, coste y memoria valorados. Este aspecto no se ha contemplado durante la realización del presente TFG al no haberse planteado como objetivo conseguir una detección de un amplio número de gestos realizados con la mano, sino la posibilidad de generar un modelo para el reconocimiento directamente en la plataforma.

- Realizar una muestra de tomas más extensa, validando la funcionalidad de la plataforma con más usuarios que permitan sacar conclusiones más generales de la potencia de la plataforma *hardware/software* desarrollada.

- Desarrollar una interfaz de usuario más amigable, sin la necesidad de recurrir a un terminal y haciéndola extensible a otro dispositivo portátil que mejore la usabilidad de las funcionalidades de la plataforma.

Bibliografía

[1] D. Evans, “The Internet of Things. How the Next Evolution of the Internet Is Changing Everything,” *Cisco Internet Bus. Solut. Gr.*, 2011, Accessed: Jan. 28, 2022. [Online]. Available: https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf.

[2] S. Rawat, S. Vats, and P. Kumar, *Evaluating and exploring the MYO ARMBAND*. 2016.

[3] I. Mendez *et al.*, *Evaluation of the Myo armband for the classification of hand motions*. 2017.

[4] F. J. Dian, A. Yousefi, and S. Lim, *A practical study on Bluetooth Low Energy (BLE) throughput.* 2018.

[5] P. Visconti, F. Gaetani, G. A. Zappatore, and P. Primiceri, “Technical features and functionalities of Myo armband: An overview on related literature and advanced applications of myoelectric armbands mainly focused on arm prostheses,” *Int. J. Smart Sens. Intell. Syst.*, vol. 11, no. 1, pp. 1–25, 2018, doi: 10.21307/IJSSIS-2018-005.

[6] I. Donovan *et al.*, “MyoHMI: A low-cost and flexible platform for developing real-time human machine interface for myoelectric controlled applications,” *2016 IEEE Int. Conf. Syst. Man, Cybern. SMC 2016 - Conf. Proc.*, pp. 4495–4500, Feb. 2017, doi: 10.1109/SMC.2016.7844940.

[7] A. Ganiev, H.-S. Shin, and K.-H. Lee, “Study on virtual control of a robotic arm via a Myo armband for the self-manipulation of a hand amputee,” 2016. https://www.researchgate.net/publication/298714563_Study_on_virtual_control_of_a_robotic_arm_via_a_Myo_armband_for_the_self-manipulation_of_a_hand_amputee (accessed Jun. 14, 2022).

[8] Thalmic Labs, “TEDCAS uses the Myo armband to give surgeons gesture control,” *Medium*, 2014. https://medium.com/thalmic/tedcas-uses-the-myo-armband-to-give-surgeons-gesture-control-d24ce740e1ff (accessed Jun. 09, 2022).

[9] B. T. Website, “The Bluetooth® Low Energy Primer,” *Bluetooth*. https://www.bluetooth.com/bluetooth-resources/the-bluetooth-low-energy-primer/ (accessed Jun. 08, 2022).

[10] B. T. Website, “Intro to Bluetooth Low Energy.” https://www.bluetooth.com/bluetooth-resources/intro-to-bluetooth-low-energy/ (accessed Jun. 04, 2022).

[11] M. Arin, P. Beloqui, and G. Ubilla, “Bluetooth Low Energy Object Finder,” *Univ. ORT Uruguay*, 2017, Accessed: Jun. 05, 2022. [Online]. Available: https://dspace.ort.edu.uy/bitstream/handle/20.500.11968/3401/Material completo.pdf?sequence=-1&isAllowed=y.

[12] J. Tosi, F. Taffoni, M. Santacatterina, R. Sannino, and D. Formica, “Performance Evaluation of Bluetooth Low Energy: A Systematic Review,” *Sensors 2017, Vol. 17, Page 2898*, vol. 17, no. 12, p. 2898, Dec. 2017, doi: 10.3390/S17122898.

[13] D. Ovsiyenko, “Diseño de un sistema Bluetooth Low Energy para la medida de fuerza,” *Esc. Tècnica d’Enginyeria Telecomunicació Barcelona, Univ. Politècnica Catalunya*, pp. 1–

61, Feb. 2020, Accessed: Jun. 05, 2022. [Online]. Available: https://upcommons.upc.edu/bitstream/handle/2117/179208/TFG_DenysOvsiyenko.pdf?sequence=1&isAllowed=y.

[14] I. Microchip Technology, “Bluetooth Low Energy Physical Layer,” *2021*. https://microchipdeveloper.com/wireless:ble-phy-layer (accessed Jun. 05, 2022).

[15] J. Molina Machado, “Diseño y desarrollo de Readers Bluetooth Low Energy para IoT,” Oct. 2018. Accessed: Jun. 05, 2022. [Online]. Available: https://repositorio.upct.es/bitstream/handle/10317/7448/tfg-mac-dis.pdf?sequence=1&isAllowed=y.

[16] C. S. Viera Betancor, “Plataforma Para La Interpretación Del Alfabeto Dactilológico De La Lengua De Signos Española Basada En El Dispositivo Myo Armband,” *Univ. las Palmas Gran Canar.*, 2020, Accessed: Jan. 27, 2022. [Online]. Available: https://accedacris.ulpgc.es/jspui/handle/10553/105068.

[17] Hacker de Cabecera, “BLECTF, Capture The Flag en formato hardware basado en Bluetooth Low Energy BLE + Write-Up,” *Bluetooth*, Dec. 17, 2019. https://www.hackerdecabecera.com/2019/12/blectf-capture-flag-en-formato-hardware.html (accessed Jun. 05, 2022).

[18] R. G. Portillo Criado, J. Vicario López, and X. Vilajosana Guillen, “Estudio en detalle y evaluación de protocolo BLE Mesh,” *Univ. Oberta Catalunya*, 2018, Accessed: Jun. 06, 2022. [Online]. Available: http://openaccess.uoc.edu/webapps/o2/bitstream/10609/80565/8/ricporcriTFM0618memoria.pdf.

[19] Cisco Systems, “Security: Encryption Manager.” https://www.cisco.com/web/techdoc/wireless/access_points/online_help/eag/123-04.JA/1100/h_ap_sec_ap-key-security.html (accessed Jun. 06, 2022).

[20] A. González Vilanova, “Métodos de machine learning en estudios biomédicos,” *Univ. Politécnica Val.*, Accessed: Jun. 07, 2022. [Online]. Available: https://m.riunet.upv.es/bitstream/handle/10251/127574/González - Métodos de machine learning en estudios biomédicos.pdf?sequence=4&isAllowed=y.

[21] L. Judith Sandoval, “Algoritmos de aprendizaje automático para análisis y predicción de datos,” *Esc. Espec. en Ing. ITCA-FEPADE Rev. Tecnológica*, vol. 11, 2018, Accessed: Jun. 07, 2022. [Online]. Available: http://www.redicces.org.sv/jspui/bitstream/10972/3626/1/Art6_RT2018.pdf.

[22] T. C. Martínez Fernández, “Comparación de modelos Machine Learning aplicados al riesgo de crédito.,” *Univ. Concepción, Chile*, 2022, Accessed: Jun. 07, 2022. [Online]. Available: http://repositorio.udec.cl/handle/11594/9846.

[23] A. Alba Vega and J. F. Calle Jara, “Aplicación de técnicas de Machine Learning basado en información sísmica para profundizar la probabilidad de terremotos mediante el uso de regresión logística y redes neuronales,” *Univ. Guayaquil, Ecuador*, 2020.

[24] G. A. Mardones Baeza and R. Paredes Moraleda, “API de reconocimiento de microexpresiones faciales utilizando algoritmos de Fisherface y Support Vector Machine,” *Univ. Talca, Chile*, 2017, Accessed: Jun. 06, 2022. [Online]. Available: http://dspace.utalca.cl/handle/1950/12266.

[25] J. Martinez Heras, “Máquinas de Vectores de Soporte (SVM),” *IArtificial*, 2019.

https://www.iartificial.net/maquinas-de-vectores-de-soporte-svm/ (accessed Jun. 07, 2022).

[26] A. López Díaz, "Fundamentos matemáticos de los métodos Kernel para aprendizaje supervisado," *Univ. Sevilla*, 2018, Accessed: Jun. 07, 2022. [Online]. Available: https://idus.us.es/handle/11441/77547#.Yp8heXomuB0.mendeley.

[27] S. Maldonado and R. Weber, "Modelos de Selección de Atributos para Support Vector Machines," *Rev. Ing. Sist.*, vol. XXVI, 2012, Accessed: Jun. 06, 2022. [Online]. Available: http://www.dii.uchile.cl/~ris/RISXXVI/maldonado1.pdf.

[28] C.-C. Chang and C.-J. Lin, "LIBSVM -- A Library for Support Vector Machines," 2021. https://www.csie.ntu.edu.tw/~cjlin/libsvm/ (accessed Feb. 08, 2022).

[29] C.-C. Chang and C.-J. Lin, "LIBSVM: A Library for Support Vector Machines," Accessed: May 21, 2022. [Online]. Available: www.csie.ntu.edu.tw/.

[30] L. D. Unapanta Benavides and M. E. Benalcázar, "Reconocimiento de gestos de la mano en tiempo real basado en señales electromiográficas utilizando Myo Armband con wavelets y máquinas de vectores de soporte," *Esc. POLITÉCNICA Nac.*, 2019, Accessed: May 26, 2022. [Online]. Available: https://bibdigital.epn.edu.ec/bitstream/15000/20242/1/CD 9704.pdf.

[31] A. Bravo, V. Pérez, L. Bermeo, F. E. Arcos, D. M. Quiguanas, and J. J. Villarejo, "Protocolo para la adquisición de señales mioeléctricas de los músculos inervados por los nervios ulnar, radial, medial para una órtesis de mano," *Univ. Santiago Cali, Univ. Fed. Paraná*, pp. 1–46, 2019, Accessed: Jun. 08, 2022. [Online]. Available: https://repository.usc.edu.co/bitstream/handle/20.500.12421/2958/PROTOCOLO PARA LA ADQUISICIÓN.pdf?sequence=3&isAllowed=y.

[32] "Las Fibras Musculares y la ganancia de masa muscular: Saca el máximo partido de tu entrenamiento," *Fitness en la nube*. https://fitnessenlanube.com/fibras-musculares/ (accessed Jun. 08, 2022).

[33] J. A. Zea Guachamín, "IMPLEMENTACIÓN DE UN SISTEMA DE CLASIFICACIÓN DE GESTOS DEL BRAZO HUMANO UTILIZANDO MYO ARMBAND PARA MANDO A DISTANCIA DE UN BRAZO ROBÓTICO DE 3GDL," *Fac. Ing. Eléctrica y Electrónica*, 2016, Accessed: May 27, 2022. [Online]. Available: https://bibdigital.epn.edu.ec/bitstream/15000/19190/1/CD-8571.pdf.

[34] E. Quijas, "UNION NEUROMUSCULAR," *SlidePlayer*, 2015. https://slideplayer.es/slide/3976978/ (accessed May 29, 2022).

[35] J. Villamizar, P. M. Sc, R. Padilla, M. Estudiante, G. Cabrera, and H. Estudiante, "Brazo robótico controlado por electromiografía," *Sci. Tech. Año XVII*, vol. 52, 2012, Accessed: May 26, 2022. [Online]. Available: https://revistas.utp.edu.co/index.php/revistaciencia/article/view/8197/5075.

[36] G. B. O., E. G. Suárez, and J. F. F. B., "Reconocimiento de patrones de movimiento a partir de señales electromiográficas," *Sci. Tech.*, vol. 3, no. 26, Jan. 2004, doi: 10.22517/23447214.7045.

[37] G. Pequera, "Análisis tiempo-frecuencia de la señal de EMG en movimientos explosivos: estudio de la coordinación en el salto vertical," *Univ. la República Uruguay*, 2015, Accessed: May 25, 2022. [Online]. Available: https://www.colibri.udelar.edu.uy/jspui/bitstream/20.500.12008/8156/1/uy24-

17718.pdf.

[38] A. Sánchez, A. Tutores, L. Enrique, M. Lorente, M. Dolores, and B. Rojas, “Matriz de electrodos EMG para detección de intención de movimiento de la mano,” *Univ. Carlos III Madrid, Ing. Electrónica Ind. y Automática*, 2016, Accessed: May 24, 2022. [Online]. Available: https://e-archivo.uc3m.es/bitstream/handle/10016/28051/TFG_Alejandro_Sanchez_Anillo_2017.pdf?sequence=1&isAllowed=y.

[39] H. J. Kim, S. Lee, and D. Kim, “Arm Motion Estimation Algorithm using MYO Armband,” 2017, doi: 10.1109/IRC.2017.32.

[40] J. S. Ortega Solórzano and D. M. Toaquiza Urrea, “Desarrollo de un sistema de comunicación inclusiva entre personas con deficiencia auditiva y su entorno: prueba de concepto,” *Univ. las Fuerzas Armadas ESPE*, 2019, Accessed: May 25, 2022. [Online]. Available: http://repositorio.espe.edu.ec/xmlui/bitstream/handle/21000/21264/T-ESPE-040856.pdf?sequence=1&isAllowed=y.

[41] L. S. Vega Escobar, “Metodología basada en entrenamiento automático para el reconocimiento del movimiento individual de los dedos de la mano usando análisis de señales electromiográficos de superficie,” no. TecnoLógicas, 2018, Accessed: May 24, 2022. [Online]. Available: https://repositorio.itm.edu.co/handle/20.500.12622/293#.Yo1KUq-SlTQ.mendeley.

[42] L. Sánchez-Velasco, M. Arias-Montiel, and E. Guzmán-Ramírez, “EMG data of the Myo Armband,” vol. 1, 2019, doi: 10.17632/RWBS7645HG.1.

[43] “GitHub - thalmiclabs/myo-bluetooth: The BLE specification of the Myo armband.” https://github.com/thalmiclabs/myo-bluetooth (accessed Jun. 07, 2022).

[44] Arduino, “ArduinoBLE.” https://www.arduino.cc/reference/en/libraries/arduinoble/ (accessed Jun. 07, 2022).

[45] J. R. Patiño R., Y. C. Acevedo C., and D. H. Albarracín S., “Diseño y construcción de un prototipo de prótesus mioeléctrica de antebrazo y mano con dos grados de libertad,” *Investig. Form. en Ing.*, vol. 2, p. 190, 2018, Accessed: May 17, 2022. [Online]. Available: https://www.researchgate.net/profile/Edgar-Serna-M/publication/331385548_Investigacion_Formativa_en_Ingenieria_ed_2/links/5c76ea1992851c695046658d/Investigacion-Formativa-en-Ingenieria-ed-2.pdf#page=190.

[46] Nearlab, “Myo Gesture Control Armband.” https://nearlab.polimi.it/medical/equipment (accessed May 10, 2022).

[47] “Diagnostics - Myo.” http://diagnostics.myo.com/ (accessed Jun. 07, 2022).

[48] Arduino, “Arduino NANO 33 IoT.” https://arduino.cl/producto/arduino-nano-33-iot/ (accessed May 17, 2022).

[49] Arduino, “Arduino® Nano 33 IoT Target areas: Maker, enhancements, basic IoT application scenarios,” 2022, Accessed: May 10, 2022. [Online]. Available: https://docs.arduino.cc/static/e9c07b3610eefd3ab3feeae9d7f90cbe/ABX00027-datasheet.pdf.

[50] “Download PuTTY - a free SSH and telnet client for Windows.” https://www.putty.org/ (accessed May 20, 2022).

[51] Processing, “Welcome to Processing!” https://processing.org/ (accessed May 20, 2022).

[52] “GitHub - radzilu/Arduino-SVM.” https://github.com/radzilu/Arduino-SVM (accessed Jun. 07, 2022).

[53] Arduino, “ARDUINO ABX00031 Nano 33 BLE Sense User Manual,” 2022, Accessed: May 03, 2022. [Online]. Available: https://manuals.plus/arduino/abx00031-nano-33-ble-sense-manual.pdf.

[54] Arduino, “Arduino Nano 33 BLE Sense.” https://store.arduino.cc/products/arduino-nano-33-ble-sense?selectedStore=eu (accessed May 13, 2022).

[55] “RAM ferroeléctrica,” *Hmong*. https://hmong.es/wiki/FeRAM (accessed May 13, 2022).

[56] Spiegato, “¿Qué es la memoria ferroeléctrica? - Spiegato.” https://spiegato.com/es/que-es-la-memoria-ferroelectrica (accessed May 13, 2022).

[57] “MB85RC256V I2C FRAM,” *Adafruit*, Accessed: May 10, 2022. [Online]. Available: https://learn.adafruit.com/adafruit-i2c-fram-breakout.

[58] “OLED-128O032D-LPP3N00000 128 x 32 Graphic OLED,” *Vishay*, Accessed: May 10, 2022. [Online]. Available: https://www.vishay.com/docs/37894/oled128o032dlpp3n00000.pdf.

[59] EastRising, “OLED Display Datasheet ER-OLED0.91-1 Series,” *EastRising Technol. Co. Ltd.*, 2009, Accessed: May 10, 2022. [Online]. Available: http://www.buydisplay.com/download/manual/ER-OLED0.91-1_Series_Datasheet.pdf.

[60] “Display Oled 0.91" I2C 128*32 SSD1306,” *Naylamp Mechatronics*. https://naylampmechatronics.com/oled/391-display-oled-i2c-091-12832-ssd1306.html (accessed May 10, 2022).

[61] “Doxygen: Doxygen.” https://doxygen.nl/ (accessed Jul. 08, 2022).

[62] “Boletín Oficial de la Universidad de Las Palmas de Gran Canaria,” Las Palmas de Gran Canaria, Jul. 2021. Accessed: Jun. 13, 2022. [Online]. Available: https://www.ulpgc.es/sites/default/files/ArchivosULPGC/boulpgc/BOULPGC/boulpgc_1_julio_2021.pdf.

Pliego de condiciones

A continuación, se exponen las condiciones que se han seguido para el desarrollo del presente Trabajo Fin de Grado, indicando las diferentes herramientas *hardware*, *software* y *firmware* empleadas.

PL.1 Condiciones *hardware*

En la Tabla 11 se recogen todos los dispositivos *hardware* que se han usado para la realización de este TFG.

Tabla 11. Relación de equipos hardware utilizados

Equipo	Modelo	Fabricante/Comerciante
Ordenador portátil	IDMK4D8 Ideapad 100 Intel(R) Celeron(R) CPU N2840	*Lenovo*
Gesture Control Armband	*Myo*	*Thalmic Labs*
Arduino Nano	33 *IoT*	*Arduino*
Arduino Nano	33 BLE *Sense*	*Arduino*
Memoria FRAM	MB85RC256V	*Fujitsu*
Display OLED	0.91” 128x32 I2C	*Adafruit*

PL.2 Condiciones *software*

En este caso, en la Tabla 12, se muestran las herramientas *software* utilizadas y se especifica su versión.

Tabla 12. Relación de equipos software utilizados

Aplicación	Versión
Sistema Operativo	Microsoft Windows 10
Microsoft Office	18.2110.13110.0
Notepad++	8.4.1
PuTTY	0.77
Arduino IDE	1.9.19
Herramienta LIBSVM	3.25
Herramienta *Processing*	4.0
Mendeley	1.19.8
Lucidchart	*Online*

Myo Armband firmware	MyoFirmware.major = 1 MyoFirmware.minor = 5 MyoFirmware.patch = 1970 MyoFirmware.hardware_rev = 2 MyoFirmware.hardware_rev = myo_hardware_rev_revd

Presupuesto

Este capítulo contiene el presupuesto que recoge los gastos generados en la realización del Trabajo Fin de Grado expuesto. Para el cálculo del presupuesto total se han tenido en cuenta aspectos como el trabajo tarifado en función del tiempo dedicado, la amortización del material *hardware* y *software*, la redacción de la documentación, los derechos de visado del Colegio Oficial de Ingenieros Técnicos de Telecomunicación (COITT) y los gastos de tramitación y envío. Por último, se aplicarán los impuestos vigentes para obtener el coste total del TFG.

P.1 Trabajo tarifado por tiempo empleado

Para el trabajo tarifado por tiempo empleado se contabilizan los gatos correspondientes a la mano de obra en función del salario correspondiente a la hora de trabajo de un ingeniero Técnico de Telecomunicación. Consultando la tabla retributiva de personal contratado en proyectos de investigación elaborada por la Universidad de Las Palmas de Gran Canaria en 2021, se comprueba que el salario para una dedicación de 20 horas a la semana se corresponde con una cantidad de 725,49 euros [62]. Como la duración de la realización del TFG es de aproximadamente 4 meses, el cálculo total será:

$$725{,}49 \times 4 = 2.901{,}96\ €$$

Por tanto, el trabajo tarifado por tiempo empleado asciende a la cantidad de *dos mil novecientos un euros con noventa y seis céntimos* (2.901,96 €).

P.2 Amortización del inmovilizado material

Para el inmovilizado material se tienen en cuenta tanto los recursos *hardware* como *software* que se han empleado en la realización del presente TFG. Para estipular el coste de amortización en un período de 3 años se utiliza un sistema de amortización lineal, donde el inmovilizado material se desprecia de forma constante a lo largo de su vida útil. La cuota de amortización anual se calcula con la fórmula que se presenta a continuación.

$$Cuota\ anual = \frac{Valor\ de\ adquisición - Valor\ residual}{Número\ de\ años\ de\ vida\ útil}$$

Donde el valor residual se corresponde con el valor teórico que se supone que tendrá el elemento después de su vida útil.

Los costes de la amortización del material *hardware* se calcularán en base a los costes derivados de los primeros 4 meses, ya que este es el tiempo de duración del Trabajo Fin de Grado y es un periodo considerablemente inferior a los 3 años estipulados. En la Tabla 13 se

especifica el hardware amortizable que ha sido requerido en la realización del TFG, indicando el valor de adquisición y su amortización.

Tabla 13. Costes y amortización del hardware.

Elemento	Valor de adquisición	Amortización
Ordenador portátil Lenovo *Ideapad* 100	400 €	100 €
Myo Gesture Control Armband	143,27 €	143,27 €
Arduino Nano 33 IoT	37,99 €	37,99 €
Arduino Nano 33 BLE Sense	53,56 €	53,56 €
Memoria FRAM MB85RC256V	20,29 €	20,29 €
Display OLED 0.91”	14,10 €	14,10 €
Pulsador	0,50 €	0,50 €
Total	**669,71 €**	**369,71 €**

Finalmente, calculando la suma total de las cantidades se determina que el coste del material *hardware* asciende a *trescientos sesenta y nueve euros con setenta y un céntimos* (369,71 €).

De la misma manera, para el material *software* se tendrán en cuenta los costes derivados de los 4 primeros meses. En la Tabla 14, se muestran los elementos *software* necesarios para la realización del trabajo, su valor de adquisición y su amortización.

Tabla 14. Costes y amortización del software

Elemento	Valor de adquisición	Amortización
Sistema operativo Windows	0,00 €	0,00 €
Microsoft Office	0,00 € (*)	0,00 € (*)
Notepad++	0,00 € (**)	0,00 € (**)
Arduino IDE	0,00 € (**)	0,00 € (**)
Processing (Arduino_SVM)	0,00 € (**)	0,00 € (**)
PuTTY	0,00 € (**)	0,00 € (**)
Mendeley	0,00 € (**)	0,00 € (**)
Lucidchar	0,00 € (**)	0,00 € (**)
LIBSVM	0,00 € (**)	0,00 € (**)
Total	**0,00 €**	**0,00 €**

(*) Licencia de uso proporcionada por la ULPGC.

(**) Software libre.

Por tanto, el coste total del material *software* es de *cero euros* (0 €).

P.3 Redacción del trabajo

Una vez calculados los costes de amortización del material, se ha determinado el coste asociado a la redacción de la memoria. Para ello, se ha tenido en cuenta la fórmula que se presenta a continuación

$$R = 0{,}07 \times P \times C_n$$

donde R representa los honorarios por la redacción del trabajo, P es el presupuesto y C_n el coeficiente de ponderación en función del presupuesto. El presupuesto se calcula sumando los costes del trabajo tarifado por tiempo empleado y la amortización del inmovilizado material. Este cálculo se refleja en la Tabla 15.

Tabla 15. Presupuesto P

Concepto	Coste
Trabajo tarifado por tiempo empleado	2.901,96 €
Amortización del material *hardware*	369,71 €
Amortización del material *software*	0,00 €
Total	**3.271,67 €**

El coeficiente de ponderación para C_n para presupuestos menores de 30.050,00 € viene definido por el COITT con un valor de 1.00, por lo que el coste derivado de la redacción del Trabajo Fin de Grado es de:

$$R = 0{,}07 \times 3.271{,}67 \times 1 = 229{,}0169\ €$$

El coste de redacción del trabajo asciende a *doscientos veintinueve euros* (229,0169 €).

P.4 Derechos del visado del COITT

Por último, el COITT establece que los derechos de visado para proyectos técnicos de carácter general se calculan en base a la fórmula que se presenta a continuación

$$V = 0{,}006 \times P_1 \times C_1 + 0{,}003 \times P_2 \times C_2$$

donde P_1 es el presupuesto del proyecto, C_1 es el coeficiente reductor en función del presupuesto, P_2 es el presupuesto de ejecución material correspondiente a la obra civil y C_2 es el coeficiente reductor en función a P_2.

P_1 se calcula sumando los costes de las secciones correspondientes al trabajo tarifado por tiempo empleado, a la amortización del inmovilizado material y a la redacción del documento. El cálculo del presupuesto se refleja en la Tabla 16. Por otra parte, el coeficiente C_1tiene un valor de 1,00 para proyectos de presupuesto inferior a 30.050,00 € y el coeficiente P_2 un valor de 0,00 € al no realizarse ninguna obra.

Tabla 16. Presupuesto P1

Concepto	Coste
Trabajo tarifado por tiempo empleado	2.901,96 €
Amortización del material hardware	369,71 €
Amortización del material software	0,00 €
Redacción del trabajo	229,0169 €
Total	**3.500,6869€**

De esta manera, se puede calcular los derechos de visado V aplicando la fórmula analizada anteriormente.

$$V = 0{,}006 \times P_1 \times C_1 + 0{,}003 \times P_2 \times C_2 = 0{,}006 \times 3.500{,}6869 \times 1 = 21{,}004\ €$$

Por tanto, los derechos de visado del presupuesto ascienden a *veintiún euros* (21,004 €).

P.5 Gastos de tramitación y envío

Los gastos de tramitación y envío están estipulados en *seis euros* (6,00 €) por cada documento visado de forma telemática.

P.6 Material fungible

Además de los recursos *hardware* y *software* se han empleado otros materiales considerados fungibles. Estos materiales, así como los costes derivados de estos recursos, se recogen en la Tabla 17.

Tabla 17. Costes de material fungible

Concepto	Coste
Folios	10 €
Total	**10 €**

Los costes de material fungible ascienden a *diez euros* (10 €).

P.7 Aplicación de impuestos y coste total

Por último, es necesario aplicar el Impuesto General Indirecto Canario (IGIC) en un 7% para la realización del presente TFG. Para presentar el presupuesto final teniendo en cuenta todos los costes e impuestos se ha elaborado la Tabla 18.

Tabla 18. Presupuesto total del Trabajo Fin de Grado

Concepto	Coste
Trabajo tarifado por tiempo empleado	2.901,96 €
Amortización del material hardware	369,71 €
Amortización del material software	0,00 €
Redacción del trabajo	229,0169 €
Derechos de visado del COITT	21,004 €
Gastos de tramitación y envío	6,00 €
Costes de material fungible	10 €
Total (sin IGIC)	**3.537,6909 €**
IGIC (7 %)	**247,638363€**
TOTAL	**3.785,33€**

El presupuesto total del Trabajo Fin de Grado "Desarrollo de una plataforma de bajo coste para el reconocimiento de gestos basado en electromiografía" asciende a *tres mil setecientos ochenta y cinco euros con treinta y tres céntimos* (3.785,33 €).

Fdo.: D. Laura Castro Vega

En Las Palmas de Gran Canaria a 11 de julio de 2022