Diseño y desarrollo de un algoritmo de inteligencia artificial embebida para controlar una neuroprótesis de estimulación eléctrica funcional en rehabilitación de miembro superior

Jeisson Esteban Bravo López y Juan Diego Peña Salinas

Trabajo de Grado para Optar al Título de Ingeniero electrónico

Director

Jorge Eduardo Quintero Muñoz

Especialista en telecomunicaciones

Codirector

Jaime Guillermo Barrero Pérez

Magíster en electrónica

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones

Ingeniería Electrónica

Bucaramanga

2024

Tabla de Contenido

Pág.

Lista de Tablas

Pág.

Lista de Figuras

Pág.

Lista de Apéndices

Pág.

Resumen

Título: Diseño y desarrollo de un algoritmo de inteligencia artificial embebida para controlar una neuroprótesis de estimulación eléctrica funcional en rehabilitación de miembro superior [*]

Autor: Jeisson Esteban Bravo López y Juan Diego Peña Salinas[**]

Palabras Clave: Algoritmo, inteligencia artificial, neuroprótesis, estimulación eléctrica funcional, rehabilitación física.

Descripción:

Los accidentes cerebrovasculares pueden provocar discapacidades severas que afectan la movilidad y la calidad de vida. Para mitigar estos efectos, se han desarrollado técnicas de estimulación eléctrica funcional que emplean pulsos bifásicos para inducir contracciones musculares, y son fundamentales en el diseño de neuroprótesis que facilitan el movimiento de las extremidades.

Este proyecto presenta un algoritmo de inteligencia artificial diseñado para detectar señales de un sensor inercial ubicado en el brazo y reconocer los movimientos para controlar los pulsos eléctricos que indican apertura o cierre de la mano. Se utiliza un acelerómetro MPU6050 y un microcontrolador ESP32; el sistema se entrena para interpretar y responder a estos movimientos específicos. El proyecto abarca la creación de un conjunto de datos, el desarrollo de un algoritmo basado en redes neuronales y series temporales, y la implementación de un sistema de control de pulsos eléctricos. Además, se ha generado documentación que se encuentra disponible en un repositorio.

Las pruebas realizadas con un circuito emulador de la neuroprótesis mostraron un 94% de precisión en la clasificación, con 170 de 180 pruebas correctamente identificadas y un tiempo de respuesta de la red neuronal de 170.3 microsegundos. También se logró reducir el consumo del microcontrolador en un 5.88% al apagar los periféricos de Bluetooth y WiFi, destacando el potencial de esta tecnología para mejorar la rehabilitación post-accidente cerebrovascular.

---

[*] Trabajo de Grado

[**] Facultad de Ingenierías Fisicomecánicas. Escuela de Ingenierías Eléctrica, Electrónica y Telecomunicaciones. Ingeniería electrónica. Director: Jorge Eduardo Quintero Muñoz. Especialista de telecomunicaciones. Codirector: Jaime Guillermo Barrero Pérez. Magíster en Electrónica.

Abstract

Title: Design and development of an embedded artificial intelligence algorithm to control a functional electrical stimulation neuroprosthesis for upper limb rehabilitation [*]

Author(s): Jeisson Esteban Bravo López and Juan Diego Peña Salinas [**]

Key Words: Algorithm, artificial intelligence, neuroprosthesis, functional electrical stimulation, physical rehabilitation

Description:

Strokes can cause severe disabilities that affect mobility and quality of life. To mitigate these effects, functional electrical stimulation techniques have been developed that employ biphasic pulses to induce muscle contractions, which are fundamental in the design of neuroprosthetics that facilitate the movement of limbs.

This project presents an artificial intelligence algorithm designed to detect signals from an inertial sensor located on the arm and recognize movements to control the electrical pulses that indicate the opening or closing of the hand. An MPU6050 accelerometer and an ESP32 microcontroller are used; the system is trained to interpret and respond to these specific movements. The project encompasses the creation of a dataset, the development of an algorithm based on neural networks and time series, and the implementation of a system for controlling electrical pulses. Additionally, documentation has been generated and is available in a repository.

Tests conducted with a neuroprosthesis emulator circuit showed 94% classification accuracy, with 170 out of 180 tests correctly identified and a neural network response time of 170.3 microseconds. A 5.88% reduction in the microcontroller's power consumption was also achieved by turning off the Bluetooth and WiFi peripherals, highlighting the potential of this technology to enhance rehabilitation after a stroke.

---

[*] Degree Work

[**] Faculty of Physical-Mathematical Engineering. School of Electrical, Electronic, and Telecommunications Engineering. Electronic Engineering. Director: Jorge Eduardo Quintero Muñoz, Telecommunications Specialist. Co-Director: Jaime Guillermo Barrero Pérez, Master's in Electronics.

Introducción

Los accidentes cerebrovasculares representan un desafío significativo para la calidad de vida de millones de personas en todo el mundo. Según la Organización Mundial de la Salud, aproximadamente 15 millones de personas sufren accidentes cerebrovasculares cada año, y al menos un tercio de ellas enfrenta alguna forma de discapacidad (Buergo et al. ,2007). Entre las secuelas más comunes se encuentra la pérdida de movilidad en el miembro superior, lo que impacta profundamente en la independencia y la capacidad de realizar actividades diarias (Vega, 2019). La rehabilitación efectiva de estos pacientes es crucial para recuperar la funcionalidad y mejorar su calidad de vida.

En el contexto de la rehabilitación post-accidente cerebrovascular, se han utilizado diversas técnicas y tecnologías, como terapias físicas convencionales, dispositivos de estimulación eléctrica funcional (FES), y entrenamientos asistidos por robot (Arias, 2009). Los dispositivos FES, que aplican pulsos bifásicos para inducir contracciones musculares, han mostrado ser útiles para facilitar el movimiento de las extremidades afectadas (Williamson & Andrews, 2000). Sin embargo, la eficacia de estos dispositivos puede verse limitada por un control manual impreciso, que a menudo no se ajusta de manera óptima a las necesidades individuales de los pacientes.

Este proyecto se enfoca en el desarrollo de un algoritmo de inteligencia artificial integrado en un microcontrolador ESP32, diseñado para detectar señales de un sensor en el brazo del paciente y clasificar la intención de movimiento para generar pulsos de Estimulación Eléctrica Funcional que faciliten la apertura y cierre de la mano. El algoritmo entrenado permite al sistema interpretar

y responder a las señales del sensor, mejorando la adaptación de los pulsos eléctricos a las intenciones del usuario.

El impacto potencial de esta solución es significativo. La integración de técnicas de inteligencia artificial en neuroprótesis abre la posibilidad de diseñar dispositivos menos invasivos y más efectivos, lo que podría transformar la rehabilitación post-accidente cerebrovascular. Al ofrecer una solución más adaptable y personalizada, este enfoque tiene el potencial de optimizar la experiencia de rehabilitación, mejorar la movilidad y la autonomía de los pacientes, y contribuir positivamente a su bienestar general.

1. Objetivos

1.1 Objetivo general

Diseñar y desarrollar un algoritmo de inteligencia artificial embebida para controlar una neuroprótesis de estimulación eléctrica funcional en rehabilitación de miembro superior.

1.2 Objetivos específicos

Crear un algoritmo que interprete las señales provenientes de sensores para reconocer patrones de movimiento de la mano, específicamente relacionados con la apertura y cierre de la misma.

Aplicar técnicas de programación que permitan reducir el consumo de energía del microcontrolador, optimizando así la autonomía del sistema.

Construir un algoritmo con documentación completa y accesible en un repositorio, facilitando las mejoras continuas y la evolución del algoritmo.

Desarrollar pruebas para evaluar el rendimiento del algoritmo de inteligencia artificial en términos de precisión, sensibilidad y tiempo de respuesta.

2. Marco teórico

2.1 Estimulación eléctrica funcional (FES)

La Estimulación Eléctrica Funcional (FES) es una técnica terapéutica que utiliza pulsos eléctricos cortos para inducir contracciones en músculos paralizados. Estas contracciones pueden coordinarse para activar las articulaciones al estimular uno o varios músculos que ejercen torques sobre la articulación. La intensidad de la estimulación se ajusta para controlar el ángulo de la articulación, actuando sobre músculos que mueven la articulación en direcciones opuestas (Lynch, 2008).

La FES se aplica en una variedad de sistemas para diferentes funciones, como la apertura de la mano, el remo estacionario y el ciclismo, así como para mejorar patrones de marcha durante la caminata (Lynch, 2008). La Estimulación Eléctrica Funcional consiste en aplicar corrientes eléctricas a neuronas motoras específicas con el fin de provocar contracciones musculares. La neurona es estimulada por una serie de pulsos eléctricos breves, aplicados a través de electrodos (Thrasher, 2004). La tensión generada en un músculo estimulado eléctricamente depende tanto de la intensidad como de la frecuencia de la estimulación. La intensidad está determinada por la cantidad total de carga transferida al músculo, la cual se ve influida por la amplitud, la duración y la frecuencia del pulso, además de la forma del tren de pulsos (Lynch, 2008).

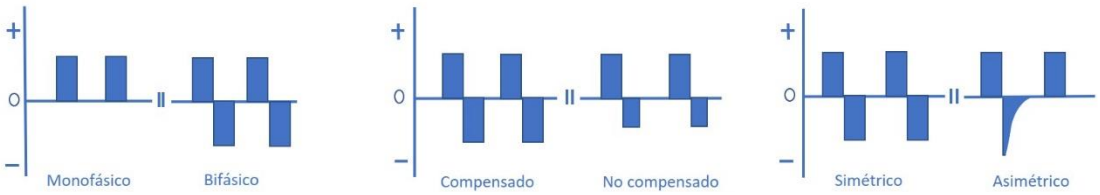
2.1.1 Parámetros en la Estimulación Eléctrica Funcional

La Estimulación Eléctrica Funcional utiliza varios parámetros que se ajustan dependiendo la aplicación específica que se desee, a continuación, se describen estos parámetros.

2.1.1.1 Forma del impulso. La forma del impulso se refiere a la geometría de la onda de corriente aplicada durante la estimulación. La forma del pulso se ajusta según el efecto deseado en el organismo (Calderón, 2019).

Figura 1.

Forma del impulso

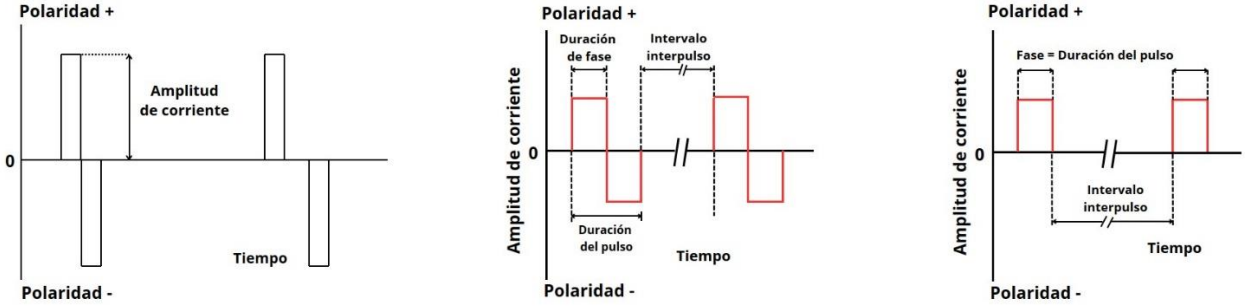


Nota. Formas de impulsos eléctricos. Tomado de (Amer Cuenca, 2011).

2.1.1.2 Duración y amplitud de los impulsos. La duración de estos pulsos depende de la respuesta neuromuscular y, en los sistemas FES, generalmente se mide en microsegundos, con valores que varían entre 100 y 400 µs. La amplitud se puede ajustar de 0 a 80 mA (Calderón, 2019).

Figura 2.

Duración y amplitud del impulso

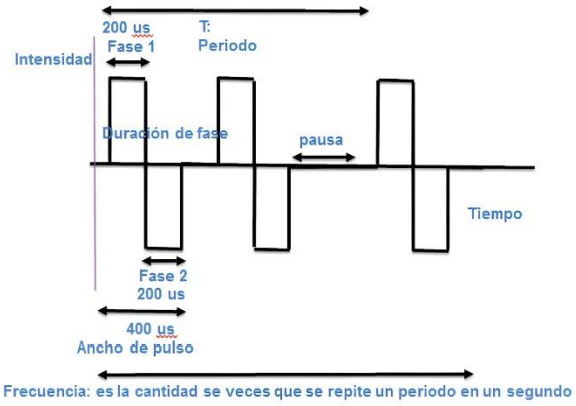


Nota. Duración y amplitud de los impulsos eléctricos. Tomado de (Fisioterapia, 2020).

2.1.1.3 Frecuencia de los impulsos. La frecuencia de pulsos en un sistema FES se refiere al número de pulsos eléctricos suministrados por segundo y se mide en Hertz (Hz). En los sistemas FES, esta frecuencia generalmente se encuentra en el rango de baja frecuencia, entre 20 y 40 Hz, aunque puede variar según la aplicación específica (Calderón, 2019).

Figura 3.

Frecuencia del impulso



Nota. Frecuencia del impulso. Tomado de (Aramayo, 2017).

2.1.2 Neuroprótesis

Las neuroprótesis motoras son dispositivos diseñados para restaurar o mejorar funciones motoras mediante la interacción con el sistema nervioso. Estos dispositivos pueden captar y decodificar señales electrofisiológicas que reflejan la intención de realizar un movimiento, (Delis, 2013).

Las neuroprótesis motoras facilitan la aplicación de Estimulación Eléctrica Funcional para activar músculos paralizados de manera secuencial y con la intensidad adecuada, permitiendo la realización de tareas funcionales (Delis, 2013).

Figura 4.

Neuroprótesis comercial

Nota. Neuroprótesis comercial para rehabilitación de mano. Tomado de (Rehabtronics, s.f.)

2.2 Inteligencia artificial

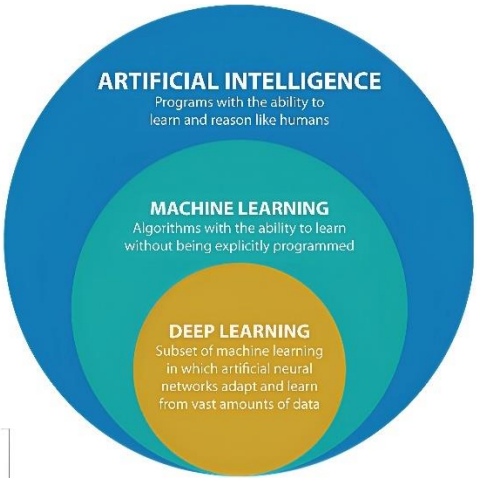
La inteligencia artificial (IA) es la habilidad de las máquinas para aplicar algoritmos, aprender a partir de datos y utilizar ese conocimiento para tomar decisiones de manera similar a cómo lo haría un ser humano (Rouhiainen, 2018).

La inteligencia artificial (IA) permite a las máquinas realizar tareas similares a las humanas, como el procesamiento del lenguaje, la identificación de objetos y la resolución de problemas. Se divide en dos tipos: IA universal, que abarca todas las capacidades humanas, e IA limitada, que destaca en tareas específicas, pero carece de habilidades en otras áreas, como una máquina que solo reconoce imágenes (Fernandez, 2021).

La inteligencia artificial (IA) abarca el aprendizaje automático (“machine learning”), que permite a las máquinas aprender de datos. A su vez, el aprendizaje profundo (“deep learning”) es un subconjunto del aprendizaje automático que utiliza redes neuronales complejas. Así, el aprendizaje profundo es parte del aprendizaje automático, y este, a su vez, es parte de la inteligencia artificial (Fernandez, 2021).

Figura 5.

Gráfico inteligencia artificial



Nota. Inteligencia artificial, “machine learning” y “Deep learning”. Tomado de (Fernandez, 2021).

2.2.1 "Machine learning"

El "machine learning" o en español aprendizaje automático tiene varias definiciones. (Géron, 2022) define al aprendizaje automático como "La ciencia (y el arte) de programar computadoras para que puedan aprender de los datos". Una definición más general sería "El aprendizaje automático es el campo de estudio que brinda a las computadoras la capacidad de aprender sin estar programado explícitamente" (Samuel, 1959).

Un algoritmo de aprendizaje automático usa datos para realizar una tarea sin estar específicamente programado para cada resultado. En lugar de ser codificado rígidamente, se ajusta automáticamente mediante la experiencia (entrenamiento) para mejorar su desempeño. Durante el entrenamiento, se le proporcionan datos y resultados deseados, permitiéndole generalizar y realizar la tarea con datos nuevos. El aprendizaje automático permite la adaptación continua, similar al aprendizaje a lo largo de la vida en los humanos (El Naqa, 2015).

2.2.1.1 Tipos de sistemas de "machine learning". Los sistemas de aprendizaje automático se clasifican según si requieren supervisión humana, aprenden de manera incremental o por lotes, y si comparan datos conocidos o crean modelos predictivos. El aprendizaje supervisado usa datos etiquetados para tareas como clasificación y regresión. En contraste, el aprendizaje no supervisado no utiliza datos etiquetados, permitiendo que el sistema descubra patrones por sí mismo. El aprendizaje semisupervisado combina datos etiquetados y no etiquetados, mientras que el aprendizaje por refuerzo involucra un agente que interactúa con su entorno, eligiendo acciones para maximizar recompensas a través de la experimentación y la mejora continua de su política de decisión (Géron, 2022).

2.2.2 Clasificación con "Machine learning"

La clasificación es un tipo de problema de aprendizaje supervisado en el que el objetivo es asignar una etiqueta a una entrada basada en sus características. Por ejemplo, clasificar correos electrónicos como "spam" o "no spam", o identificar la especie de una flor basada en medidas como el largo y ancho de los pétalos (Géron, 2022).

2.2.2.1 Métricas para Clasificación con "Machine learning". En la evaluación de modelos de clasificación mediante algoritmos de "Machine Learning", se emplean diversas métricas para medir el desempeño del modelo. Entre las más utilizadas se encuentran la precisión ("precision"), la sensibilidad ("recall"), la exactitud ("accuracy") y el F1-Score. Estas métricas permiten evaluar la calidad de las predicciones del modelo, considerando tanto su capacidad para identificar correctamente las instancias positivas como su habilidad para evitar falsos positivos y falsos negativos. (Géron, 2022).

2.3 Inteligencia artificial embebida

El uso de hardware embebido en inteligencia artificial permite almacenar algoritmos de programación diseñados para realizar tareas específicas y en tiempo real. Estos sistemas embebidos integran la inteligencia artificial directamente en el hardware, optimizando el procesamiento de datos y la toma de decisiones

Estos sistemas permiten realizar múltiples tareas simultáneamente y facilitan el desarrollo de aplicaciones en áreas como la robótica, la visión artificial y el control de sistemas avanzados (Ramos, 2016).

La IA embebida ofrece varios beneficios sobre la IA en la nube. Mejora la eficiencia del ancho de banda al reducir la necesidad de transmisión de datos, optimiza la eficiencia energética al realizar cálculos localmente y disminuye la latencia al procesar datos en el dispositivo, lo cual es crucial para aplicaciones sensibles al tiempo. Además, refuerza la privacidad al mantener los datos en el dispositivo (Lin, 2023).

2.3.1 Integración con Sensores y Microcontroladores

La integración de sensores con microcontroladores en el ámbito de la inteligencia artificial embebida permite la recopilación y procesamiento de datos en tiempo real para aplicaciones como la robótica, dispositivos portátiles y sistemas de control autónomos (Zhang, 2023). Los sensores proporcionan datos críticos que los microcontroladores procesan a través de algoritmos de IA, lo que optimiza la toma de decisiones en entornos locales sin necesidad de depender de una infraestructura de computación en la nube. Esta integración es esencial para aplicaciones donde la latencia, la eficiencia energética y la autonomía son factores clave en el diseño de sistemas embebidos.

2.3.1.1 Uso de microcontroladores en IA embebida. Los microcontroladores son ideales para ejecutar algoritmos de IA debido a su bajo consumo de energía, tamaño compacto y capacidad de procesamiento en tiempo real, al combinarse con técnicas de inferencia y aprendizaje

automático optimizadas para hardware, los microcontroladores permiten implementar redes neuronales y otros modelos de IA directamente en el dispositivo, sin necesidad de conectarse a la nube, lo que reduce la latencia y mejora la seguridad y la eficiencia energética del sistema (Shi, 2020).

2.3.1.2 Aprendizaje automático para ESP32. En cursos en línea se ofrece información sobre aprendizaje automático enfocado en microcontroladores. El curso de Udemy (Aprendizaje Automático para Arduino, 2019) explica cómo implementar algoritmos de aprendizaje automático utilizando Arduino y ESP32, empleando C++ en el entorno Arduino IDE y Python en IDLE Python.

2.3.1.3 Adquisición de señales inerciales. La adquisición de señales inerciales es fundamental en aplicaciones de inteligencia artificial embebida, como la robótica y la navegación autónoma. Los sensores inerciales, como acelerómetros y giroscopios, proporcionan datos precisos sobre movimientos y orientaciones, lo que permite el desarrollo de modelos de IA capaces de interpretar y reaccionar ante cambios en el entorno (Mascret, 2022).

3. Metodología

A lo largo de esta sección se describen los diferentes componentes de la metodología empleada, desde la adquisición de los datos, la extracción de características y el preprocesamiento de estas, hasta el diseño, entrenamiento e implementación del algoritmo de inteligencia artificial. También se abordará el proceso de integración del sistema con el controlador de la neuroprótesis

FES, así como los criterios utilizados para la activación del dispositivo. En el siguiente diagrama se describe el proceso que se lleva a cabo para el desarrollo del proyecto.

Figura 6.

Diagrama de metodología

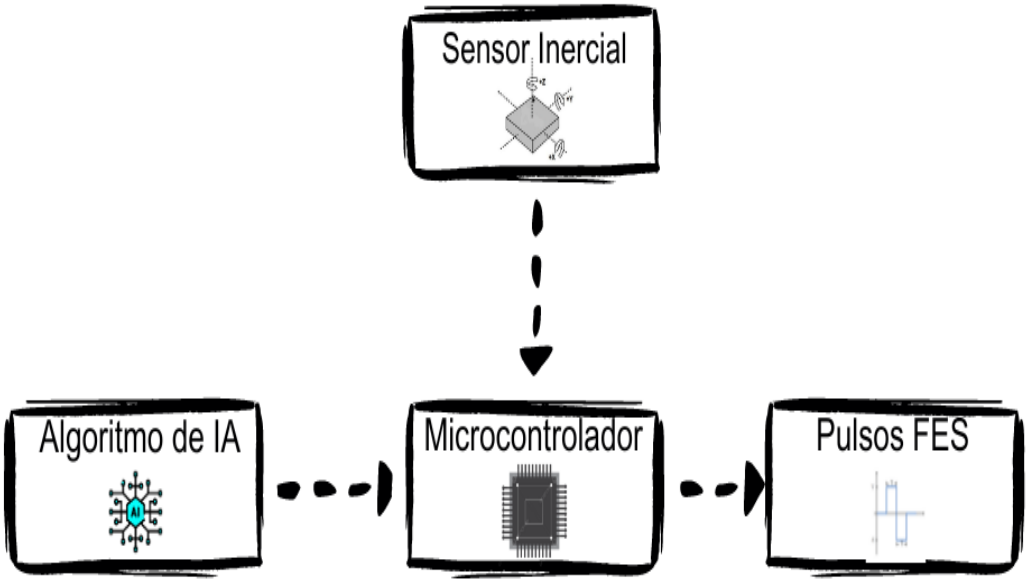
Nota. Metodología para el desarrollo del proyecto. Recursos propios

3.1 Contextualización del problema

Este proyecto busca desarrollar un algoritmo de inteligencia artificial embebida para controlar una neuroprótesis FES. El propósito es interpretar las intenciones de movimiento de una persona a partir de señales inerciales y convertir esas intenciones en instrucciones que activen los pulsos eléctricos de la neuroprótesis, permitiendo la apertura o cierre de la mano del usuario. En la siguiente figura se muestra el esquema general del sistema.

Figura 7.

Diagrama general del sistema



Nota. Diagrama general del sistema con la incorporación de las etapas correspondientes. Recursos propios.

Las entradas del modelo consistirán en series temporales de datos inerciales. Estas señales serán procesadas para extraer información útil que permita clasificar los movimientos. Las salidas

del modelo indican la clasificación realizada, especificando si los datos de entrada generan una acción de apertura o cierre de la mano.

El problema abordado es de clasificación, donde el algoritmo será responsable de identificar si el movimiento registrado corresponde a una apertura o cierre de la mano a partir de las características extraídas de los datos.

3.2 Revisión del estado del arte

En esta sección se consultan los estudios previos (Aprendizaje Automático para Arduino, 2019) con el objetivo de determinar los requisitos necesarios para el diseño del sistema. Con base en lo anterior, se determinan los componentes necesarios para el proyecto. En primer lugar, se seleccionan los lenguajes y entornos adecuados para el desarrollo del algoritmo, tal como se muestra en el marco teórico (2.3 Inteligencia artificial embebida, p. 15). A continuación, se definen los componentes necesarios para generar los pulsos de la estimulación eléctrica.

3.3 Definición de requerimientos

Teniendo en cuenta lo anterior se definen los componentes que se van a utilizar para la realización del proyecto.

Lenguajes de programación:

En el campo del “machine learning”, se utilizan diversos lenguajes de programación, como: Java Script, C++ y Python. En este caso, se opta por Python debido a su extensa colección de bibliotecas especializadas en aprendizaje automático (como TensorFlow, Keras y Scikit-learn), su

fuerte soporte comunitario y la disponibilidad de herramientas que facilitan la creación, prueba y despliegue de modelos de “machine learning”.

Entornos de desarrollo:

Para el desarrollo del algoritmo de inteligencia artificial, se elige utilizar el IDLE de Python. Adicionalmente, se emplea el IDE de Arduino para embeber la red neuronal en el microcontrolador seleccionado. La elección de estos entornos se basa en la experiencia adquirida en su uso y en el respaldo proporcionado por estudios previos consultados en el marco teórico.

Componentes de Hardware:

Sensor inercial

En la tabla 1 se describen las características y precio de los sensores inerciales comerciales.

Tabla 1.

Sensores inerciales

Nombre	Imagen	Características	Precio	Referencia
BNO055		• Acelerómetro, giroscopio y magnetómetro. • 9 ejes • Alimentación 2.4V a 3.6V	*$70395*	*BNO055*

Nombre	Imagen	Características	Precio	Referencia
MPU-6050		• Acelerómetro, giroscopio. • 6 ejes • Alimentación 2.375V a 3.46V	*$9999*	*MPU6050*
MPU-9250		• Acelerómetro, giroscopio y magnetómetro. • 9 ejes Alimentación 2.4V a 3.6V	*$44000*	*MPU9250*
BMI160		• Acelerómetro, giroscopio. • 6 ejes • Alimentación 1.71V a 3.6V	*$22990*	*BMI160*

Nota. Esta tabla muestra los diferentes sensores inerciales con sus respectivas características.

Se opta por utilizar el sensor MPU-6050 debido a que sus mediciones de aceleración y giroscopio son suficientes para la aplicación prevista, además de ser la opción más económica y de fácil acceso.

Tarjeta de desarrollo

En la tabla 2 se describen las características y precio de las tarjetas de desarrollo disponibles comercialmente.

Tabla 2.

Tarjeta de desarrollo

Nombre	Imagen	Características	Precio	Referencia
Raspberry Pi Pico W		• Procesador doble núcleo RM Cortex-M0+ • Conectividad: Wi-Fi • Alimentación 1.8V a 5.5V • Programación usando Python.	*$70395*	*Raspberry Pi Pico*
ESP32 WROOM-32		• Procesador doble núcleo Xtensa® 32-bit LX6 • Conectividad: Wi-Fi y Bluetooth. • Alimentación 5VDC	*$28500*	*ESP-32*

Nombre	Imagen	Características	Precio	Referencia
		• Compatibilidad con las librerías del IDE de arduino.		
ESP8266		• Procesador Tensilica L106 • Conectividad: Wi-Fi y Bluetooth. • Alimentación 5VDC • Compatibilidad con las librerías del IDE de arduino.	*$12800*	*ESP8266*

Nota. Esta tabla muestra las diferentes tarjetas de desarrollo.

Se elige la tarjeta de desarrollo ESP32 porque ofrece un balance entre capacidad de procesamiento y facilidad de uso en el IDE de Arduino. A diferencia de la Raspberry Pi, que se programa principalmente en Python, y de la ESP8266, que cuenta solo con un núcleo, la ESP32 destaca por su procesador de doble núcleo y su compatibilidad con Arduino, cumpliendo con ambos criterios de selección.

3.4 Adquisición de datos y extracción de características

Para la adquisición de datos, se evaluaron dos posibilidades. En primer lugar, se revisaron las bases de datos existentes, encontrando conjuntos de datos con señales inerciales tomadas con diversos sensores, incluyendo el MPU6050, que recopilan diferentes tipos de movimientos. La segunda opción considerada fue la recolección de señales inerciales de forma independiente. Esta última opción se consideró más adecuada debido a que los estudios previos consultados también realizaron la recolección de datos directamente y las cantidades requeridas no eran grandes.

Para la recolección de datos, se emplea un código en Arduino (Anexos) basado en estudios previos. Este código lee información del sensor MPU6050, el cual mide el movimiento en tres direcciones (X, Y y Z). Al presionar un botón, el código recopila datos del sensor durante un período determinado de tiempo, generando una serie temporal, que es un conjunto de mediciones de aceleración en los ejes X, Y y Z recopiladas a intervalos regulares. De esta serie temporal se extraen los datos de aceleración.

A partir de esta serie temporal, se calculan tres características clave: el Valor Medio Absoluto (MAV) el cual se calcula sumando el valor absoluto de la aceleración dividido en la cantidad de muestras, esto se puede expresar de la siguiente manera: $\mathrm{MAV} = (\mathrm{Suma}|a_i|)/\mathrm{N}$, el Valor Medio Cuadrático (RMS) se obtiene calculando la raíz cuadrada de la suma de los cuadrados de las mediciones de aceleración, dividida entre el número total de muestras. Se puede representar como: $\mathrm{RMS} = [(\mathrm{Suma}|a_i|)^2/\mathrm{N}]^{0.5}$ y la Longitud de Onda (WL) se calcula sumando las diferencias absolutas entre las mediciones de aceleración consecutivas, esto se puede expresar como: $\mathrm{WL} = \mathrm{Suma}|a_i - a_{i-1}|$. Estas características fueron seleccionadas específicamente para entrenar el

algoritmo de IA que controlará una neuroprótesis, ya que permiten representar el movimiento de manera eficiente. El MAV ofrece una medida promedio de la magnitud del movimiento en cada eje, lo cual es útil para identificar la intensidad general del movimiento. El RMS evalúa la energía de la señal, lo que ayuda a capturar movimientos más vigorosos o rápidos, esenciales para una respuesta precisa de la prótesis. Por último, la WL mide la variabilidad de la señal, permitiendo detectar cambios bruscos o patrones irregulares que son relevantes para el control fino de la prótesis.

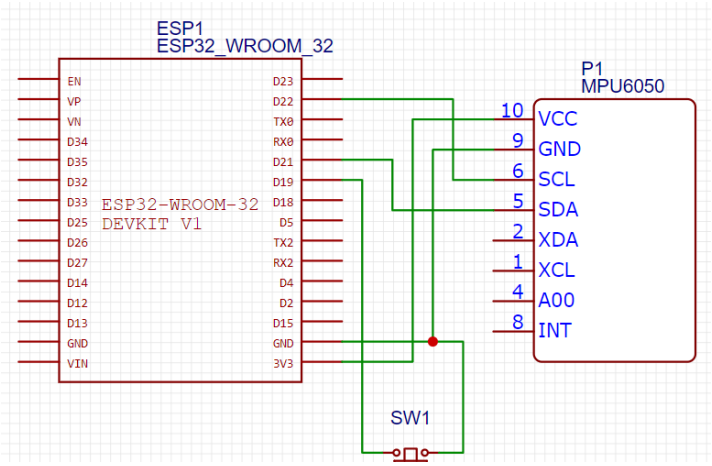
Las características seleccionadas permiten capturar diferentes aspectos del movimiento, como su magnitud, energía y variabilidad, proporcionando una representación más eficiente para el algoritmo. Estas medidas resumen la información clave de forma compacta y relevante, facilitando que el modelo aprenda los patrones de movimiento de manera más precisa y rápida.

Estas tres características resumen los datos de manera compacta pero informativa, facilitando que el algoritmo de IA aprenda los patrones de movimiento necesarios para controlar la neuroprótesis de manera precisa y en tiempo real.

Una vez calculadas las características, el código las envía por el puerto serie. Durante este proceso, un LED se enciende para indicar que el procesamiento está en curso y se utiliza una función que evita la lectura múltiple del botón. Finalmente, un código en Python (Anexos) recoge los datos, los almacena en un archivo utilizando NumPy y genera gráficos para visualizar la evolución de cada característica. A continuación, se presenta el esquemático del circuito diseñado para la recolección de datos.

Figura 8.

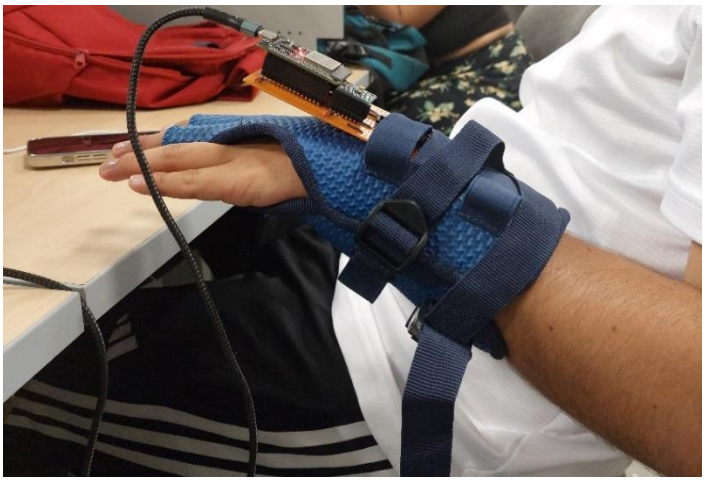
Esquemático recolección de datos



Nota. Esquema del circuito para realizar la recolección de datos. Recursos propios

Figura 9.

Sujeto usando el sistema para la recolección de datos



Nota. Sujeto usando el sistema que va a recolectar los datos para la construcción de la base de datos. Recursos propios

Descripción del "dataset":

Para crear la base de datos, se organizan tres conjuntos de datos. El primer conjunto almacena los movimientos del brazo que indican la apertura de la mano; el segundo, los movimientos que indican el cierre de la mano; y el tercero registra movimientos que no están relacionados con ninguna acción específica. El objetivo de esta última categoría es evitar falsos positivos al detectar movimientos que no corresponden a las acciones de apertura o cierre. En este caso, el movimiento de apertura de la mano se asocia con estirar el brazo (simulando un intento de alcance), mientras que el cierre de la mano se activa al mover el brazo ligeramente en dirección horizontal, los movimientos que no se asocian a ninguna acción corresponden a la elevación y giro del brazo. En total, se recolectaron 300 movimientos de dos sujetos para el entrenamiento del modelo, y 60 movimientos adicionales para el conjunto de prueba, los cuales se recolectaron de un tercer sujeto, esta cantidad de datos se recolectó considerando los estudios previos consultados. La distribución de estos movimientos se detalla en la tabla a continuación.

Tabla 3.

Descripción del "dataset"

Datos de entrenamiento					
Sujeto	**Movimiento**	**Brazo Izquierdo**	**Brazo derecho**	**Movimientos totales**	**Total por sujeto**
Sujeto 1	Alcance	25	25	50	150
	Horizontal	25	25	50	
	Elevación y giro	50		50	
Sujeto 2	Alcance	25	25	50	150

	Horizontal	25	25	50	
	Elevación y Giro	50		50	
Movimientos totales por brazo		150	150	300	
Datos de prueba					
Sujeto 3	Alcance	10	10	20	60
	Horizontal	10	10	20	
	Elevación y Giro	20		20	
Movimientos totales por brazo		30	30	60	

Nota. Esta tabla muestra el "dataset" con los movimientos registrados.

Tabla 4.

Distribución del "dataset"

Categoría	**Total Movimientos**	**Porcentaje**	**Clase 0**	**Clase 1**	**Clase 2**
Entrenamiento	240	66.66%	80	80	80
Validación	60	16.67%	20	20	20
Testeo	60	16.67%	20	20	20

Total	360	100%	120	120	120

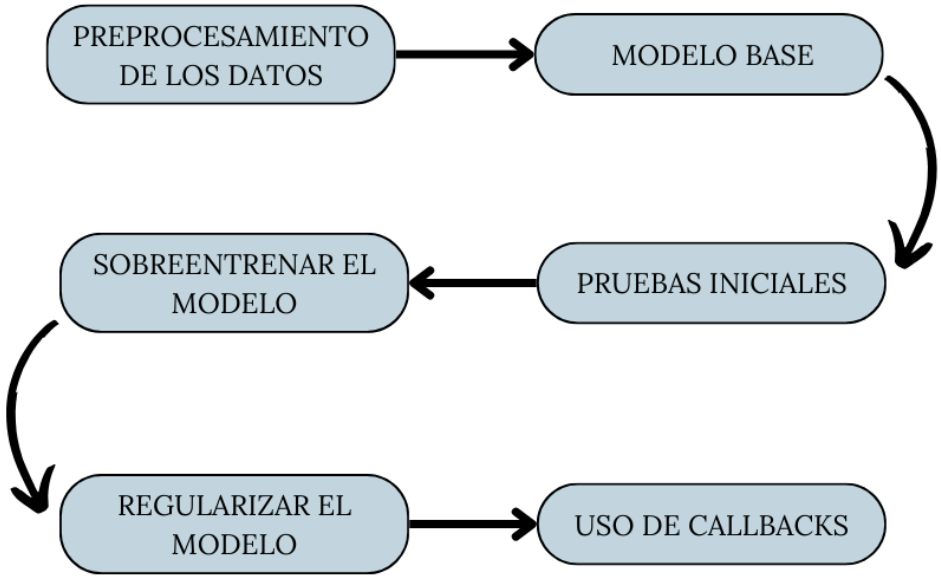
Nota. Esta tabla muestra el "dataset" con los movimientos registrados.

3.5 Preprocesamiento de los datos y diseño del modelo de IA

Una vez finalizada la recolección de datos, se procede con el preprocesamiento de los datos y el desarrollo del modelo de inteligencia artificial. El siguiente esquema detalla los pasos a seguir en esta fase.

Figura 10.

Desarrollo del modelo



Nota. Metodología para el desarrollo del modelo de IA. Recursos propios

Preprocesamiento de los datos

Como se detalló anteriormente, el desarrollo del modelo se lleva a cabo en el entorno IDLE de Python. Se comienza importando las bibliotecas necesarias para el aprendizaje automático y cargando el conjunto de datos correspondiente. El proceso de preparación de datos incluye varios pasos: primero, se concatenan los datos de diferentes clases en un único conjunto de entrenamiento y prueba. A continuación, se generan etiquetas para cada clase, las cuales se convierten al formato "one-hot" para facilitar la clasificación multiclase. Después, se normalizan los datos con "StandardScaler" (herramienta en "machine learning" que normaliza características, transformando los datos para que tengan media cero y desviación estándar uno) para estandarizar las características y mejorar la eficiencia del entrenamiento. Finalmente, el conjunto de entrenamiento se divide en subconjuntos de entrenamiento y validación.

Modelo base

En los estudios previos se propone una red neuronal que consiste en un modelo que consta de dos capas densas. La red recibe cuatro entradas, la primera capa tiene 5 neuronas y utiliza la función de activación tangente hiperbólica (tanh). La segunda capa tiene 3 neuronas que corresponden a las salidas y usa la función de activación "softmax", generando una salida de clasificación multiclase. El modelo se compila con el optimizador de descenso de gradiente estocástico (SGD), la función de pérdida es "categorical_crossentropy", y evalúa el rendimiento usando el "Accuracy" (predicciones correctas sobre el total de predicciones) (Tresadern et al. ,2006).

Para probar este modelo se debe adecuar la entrada y la salida. Las entradas del modelo corresponden a las características descritas en la sección (3.4 Adquisición de datos y extracción de características) es decir tres características por cada eje para un total de 9 datos de entrada. La capa de salida en este caso coincide con la del modelo base, así como la función de pérdida.

Pruebas iniciales

Se lleva a cabo un entrenamiento con un conjunto de datos de 30 movimientos por cada clase seleccionados de manera arbitraria, sin una regla específica, siguiendo una recomendación común en proyectos de “Machine Learning”.

Sobreentrenar el modelo

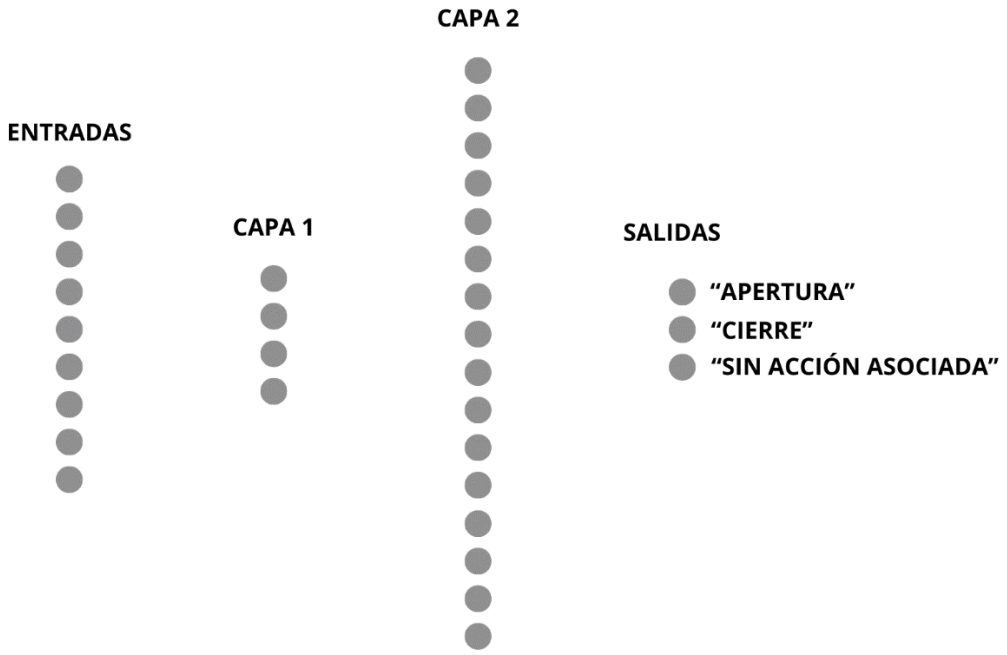
Con el set de datos descrito anteriormente se realiza un entrenamiento para verificar el sobreajuste del modelo, para esto se usa 5000 épocas en el entrenamiento con un set de validación del 20%. Al verificar el modelo se evidencia que el “accuracy” en el entrenamiento es del 100% mientras que en validación es del 94%, y la pérdida es de 0.0545 para entrenamiento y 0.15 para validación. Teniendo en cuenta los resultados se puede observar que la pérdida en el conjunto de datos de validación deja de disminuir en cierto punto, lo que sugiere que el modelo se sobreentrenó como se esperaba.

Modelo propuesto

Con base en esto se diseña un nuevo modelo que consta de tres capas densas: la primera con 4 neuronas, la segunda con 16 neuronas, y la capa de salida con 3 neuronas que corresponden a las clases de “Apertura”, “Cierre” y “Sin acción asociada” como se ilustra en la figura 11.

Figura 11

Estructura de la red neuronal propuesta

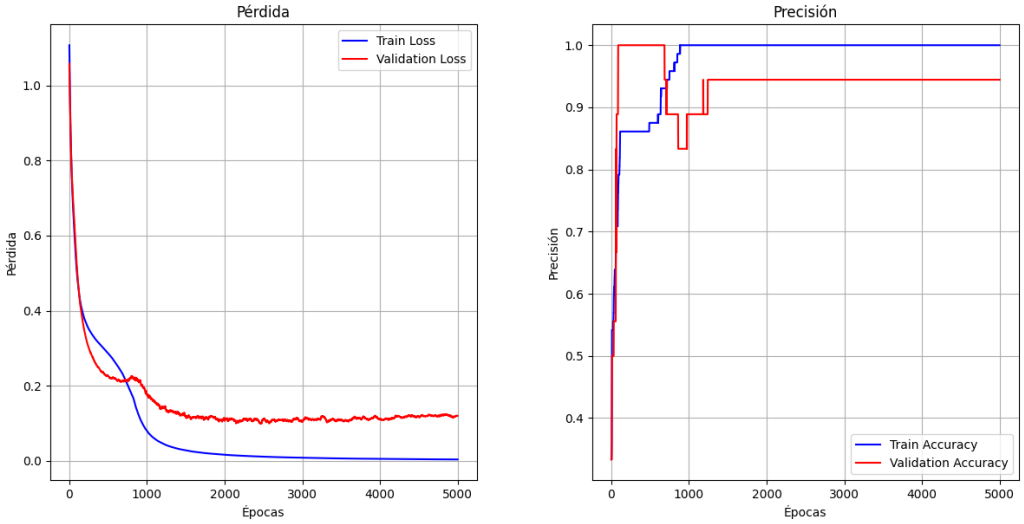


Nota. Esquema de la red neuronal con las entradas y número de neuronas para cada capa. Recursos propios

Además, se ha sustituido el optimizador por Adam y se ha implementado la función de activación "ReLU" para volver a realizar la prueba de sobreentrenamiento con un modelo propio.

Figura 12.

Gráfica precisión y pérdida



Nota. Gráficas de prueba de sobreentrenamiento del modelo. Recursos propios

Los resultados para este modelo son similares, el modelo se sobreentrena, los datos de "accuracy" son de 100% para entrenamiento y 94% para validación igual que el modelo base, sin embargo, la pérdida en este caso es menor con 0.0033 para entrenamiento y 0.1193 para validación.

Regularizar el modelo

En la red neuronal, se aplica regularización L2 con un valor de 0.01 a las capas densas intermedias. Además, se utiliza regularización L1 y L2 en la capa de salida con valores de 0.01 para cada tipo de penalización

Configuración de callbacks

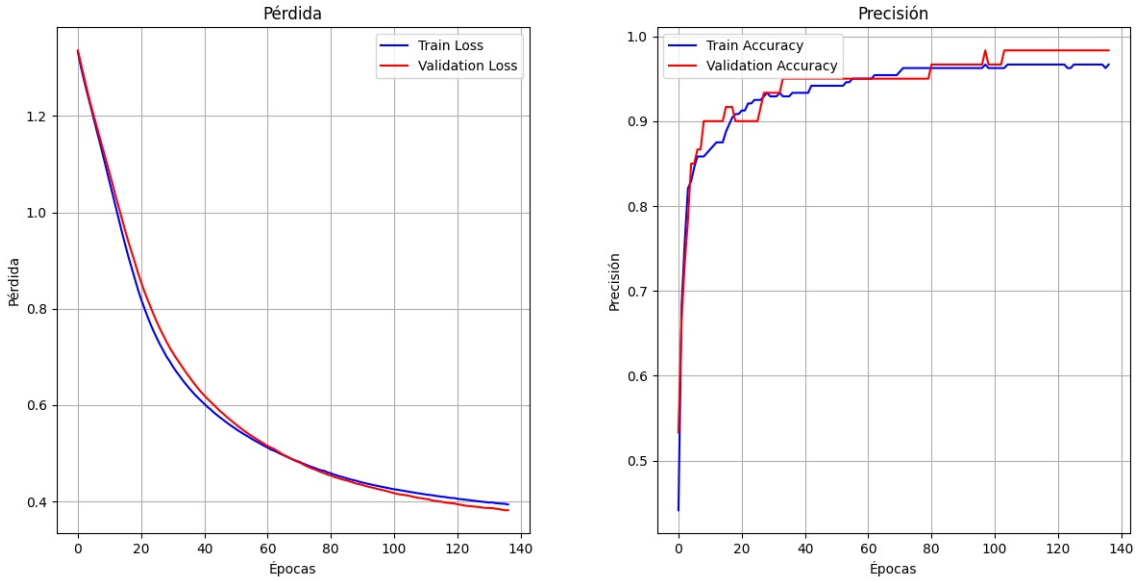
La red neuronal utiliza dos callbacks: "EarlyStopping", configurado con monitor='val_loss' y patience=1, detiene el entrenamiento si la pérdida de validación no mejora en 1 época y restaura los mejores pesos. ReduceLROnPlateau se configura con monitor='val_loss', factor=0.5,

patience=10, y min_lr=1e-6 para reducir la tasa de aprendizaje a la mitad si la pérdida se estanca durante 10 épocas, optimizando así la convergencia del modelo.

Luego de regularizar el modelo y usar callbacks se entrena el modelo con el "dataset" completo, adicionalmente se incluye el set de datos de prueba. A continuación, se muestra la gráfica de Precisión y pérdida para el modelo.

Figura 13.

Gráfica precisión y pérdida modelo ajustado



Nota. Gráficas de pérdida y precisión del modelo. Recursos propios

En este caso, el "accuracy" fue del 95.83% en el conjunto de datos de entrenamiento y del 98.33% en el conjunto de validación. La pérdida fue de 0.3491 para el conjunto de entrenamiento y de 0.3224 para el de validación. Para evaluar el rendimiento del modelo, se calcularon las

métricas de precisión, “recall”, “F1 score” y exactitud (“accuracy”) todo sobre el conjunto de prueba.

Tabla 5.

Métricas de rendimiento

	TP	**FP**	**FN**	**RECALL**	**PRECISION**	**F1 SCORE**
CLASE 0	20	1	0	1.00	0.95	0.97
CLASE 1	20	0	0	1.00	1.00	1.00
CLASE 2	19	0	1	0.95	1.00	0.97
ACCURACY				0.98		

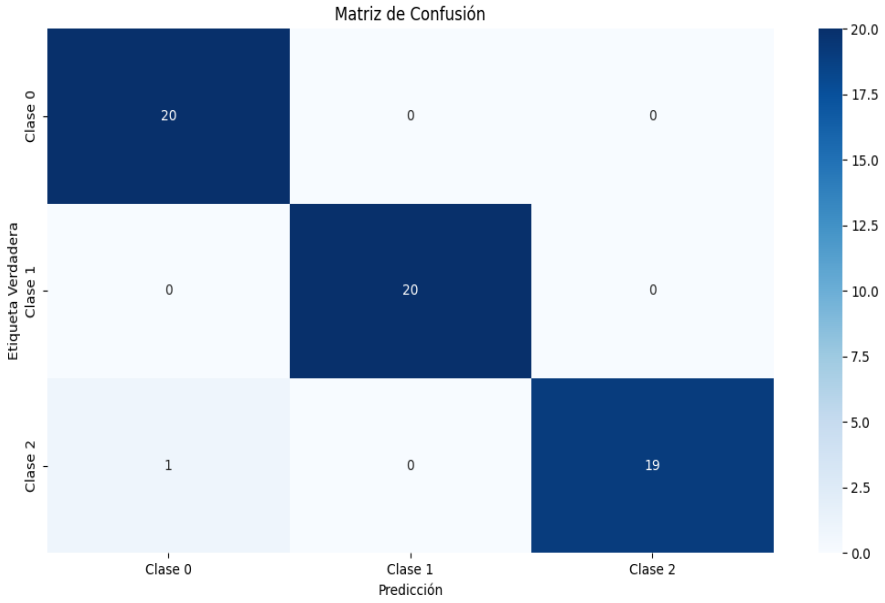
Nota. Esta tabla muestra las métricas de rendimiento del modelo.

Se emplea el método de “argmax” para clasificar instancias, eligiendo la clase con la probabilidad más alta entre todas las posibles. Cada clase tiene su propia probabilidad, y la clasificación se basa únicamente en seleccionar la que presenta el valor más elevado.

Adicionalmente se muestra la matriz de confusión sobre el conjunto de prueba.

Figura 14.

Matriz de confusión



Nota. Matriz de confusión para el conjunto de datos de prueba. Recursos propios

Se puede evidenciar que de los 60 datos de prueba se clasificaron bien 59, el dato mal clasificado corresponde a la clase 2 (Movimiento sin acción asociada) que el algoritmo clasificó como clase 0 (Apertura).

3.6 Integración del sistema de control FES

En esta etapa, el objetivo es integrar el modelo de “machine learning” con el sistema FES mediante la inferencia de la red neuronal. Para ello, se utiliza un código en Python, basado en estudios previos (Aprendizaje Automático para Arduino, 2019), que genera código en C++ para la implementación en el Arduino IDE. Primero, el código crea las declaraciones de las matrices de

pesos y sesgos con precisión decimal. Luego, genera el código para el escalado de datos utilizando los parámetros de preprocesamiento. Finalmente, produce el código necesario para las operaciones de la red neuronal, incluyendo cálculos y funciones de activación.

Luego de esto se importan las funciones correspondientes en el código de la red neuronal y se imprimen los resultados, en la siguiente figura se visualizan los resultados.

Figura 15.

Resultados red neuronal

```
//////////////////////////////// Variables Red Neuronal ////////////////////////////////
double a0[9];
double W1[4][9] = {{-0.086,-0.43,0.331,-0.426,-0.393,-0.143,-0.283,-0.637,-0.039},{0.5,-0.041,-0.511,0.068,0.387,-0.434,0.525,-0.121,-0.163},{-0.167,-0.093,
-0.043,-0.029,0.058,0.112,-0.314,1.363,-0.464},{-0.37,0.319,0.115,-0.383,0.617,0.339,-0.236,-0.118,0.591}};
double a1[4];
double W2[16][4] = {{0.666,0.074,-0.201,-0.057},{0.248,0.04,-0.087,0.052},{-0.081,0.555,-0.066,-0.101},{0.068,0.028,-0.033,-0.024},{0.383,-0.077,-0.12,-0.17
},{-0.003,0.0,-0.003,0.045},{0.149,-0.021,-0.004,-0.031},{-0.112,0.44,0.071,0.61},{-0.09,0.56,-0.079,-0.108},{0.536,-0.088,-0.136,-0.222},{-0.187,0.5,0.331,
0.627},{0.0,0.0,-0.0,-0.0},{0.536,0.209,-0.198,-0.006},{0.619,-0.117,0.22,-0.271},{-0.165,-0.093,0.986,-0.124},{-0.065,-0.06,0.424,-0.184}};
double a2[16];
double W3[3][16] = {{-0.304,-0.062,-0.073,-0.0,-0.0,0.0,-0.0,-0.095,-0.1,-0.001,0.001,0.0,-0.381,-0.0,0.697,0.35},{0.412,0.063,-0.001,-0.0,0.285,0.0,0.0,-0.
434,-0.0,0.431,-0.638,0.0,0.223,0.379,-0.505,-0.0},{0.0,0.0,0.318,-0.0,-0.0,-0.0,0.0,0.305,0.312,-0.004,0.249,0.0,-0.0,-0.406,-0.391,-0.153}};
double a3[3];
double b1[4]= {0.462,0.346,0.561,0.39};
double b2[16]= {0.403,0.123,0.198,0.215,0.382,-0.132,0.176,0.148,0.222,0.398,0.277,-0.026,0.419,0.412,0.166,0.175};
double b3[3]= {0.006,0.046,-0.03};
double aux = 0.0;
////////////////////////////////////////////////////////////////


//////////////////////////////// Preprocesamiento Red Neuronal ////////////////////////////////
double mean[9]={1.844,1.033,7.941,2.212,1.176,8.121,10.044,6.155,14.587};
double dstd[9]={1.934,0.518,1.36,2.351,0.592,1.03,7.054,3.362,4.395};
////////////////////////////////////////////////////////////////////////////////////////////////


//////////////////////////////// Estructura Red Neuronal ////////////////////////////////
for(int i = 0 ; i<4; i++ ) {aux=0.0;for(int j = 0 ; j <9 ; j++ ) { aux=aux+W1[i][j]*a0[j];} a1[i]=relu(aux+b1[i]);}
for(int i = 0 ; i<16; i++ ) {aux=0.0;for(int j = 0 ; j <4 ; j++ ) { aux=aux+W2[i][j]*a1[j];} a2[i]=relu(aux+b2[i]);}
double aux1 = 0;
for(int i = 0 ; i<3; i++ ) {aux=0.0;for(int j = 0 ; j <16 ; j++ ){ aux=aux+W3[i][j]*a2[j];} a3[i]=(aux+b3[i]);aux1=aux1+exp(a3[i]);}
double minimo = 0.0;
int classes = 0;
for(int i = 0;  i<3; i++){a3[i] = exp(a3[i])/aux1;if(a3[i]>minimo){minimo=a3[i];classes=i;}}
////////////////////////////////////////////////////////////////////////////////////////////
```

Nota. Impresión del código escrito en C++ para la extracción de variables, preprocesamiento y estructura de la red neuronal. Recursos propios

Una vez escrita la red neuronal en el lenguaje adecuado para el microcontrolador, se incorpora en el Arduino IDE para realizar la inferencia en la ESP32. Se elabora un código de inferencia (consultar Anexos) que opera de la siguiente manera: el sistema lee continuamente los datos de aceleración del sensor. Cuando detecta un movimiento significativo, es decir, un cambio en la aceleración que excede un umbral determinado enciende un LED para indicar que se está produciendo un evento. El valor del umbral se estableció en 1.5 a través de ensayos y ajuste

mediante prueba y error. Posteriormente, el sistema extrae las características de los datos de aceleración, las normaliza para asegurar que estén en un formato adecuado para el análisis, y las pasa a través de la red neuronal embebida en la ESP32, que clasifica el tipo de movimiento basado en los datos procesados.

Después de realizar la inferencia, se deben generar los pulsos FES, según la clase predicha por la red neuronal. El sistema activa una de dos secuencias de pulsos predefinidas: una secuencia de pulsos controla los pines para realizar una acción asociada con la clasificación, como "abrir", mientras que la otra realiza una acción diferente, como "cerrar"; como se explicó previamente, la tercera clase no lleva a cabo ninguna acción específica. Tras ejecutar la secuencia de pulsos correspondiente, el LED se apaga y el sistema vuelve a su estado de monitoreo para detectar y responder a nuevos movimientos.

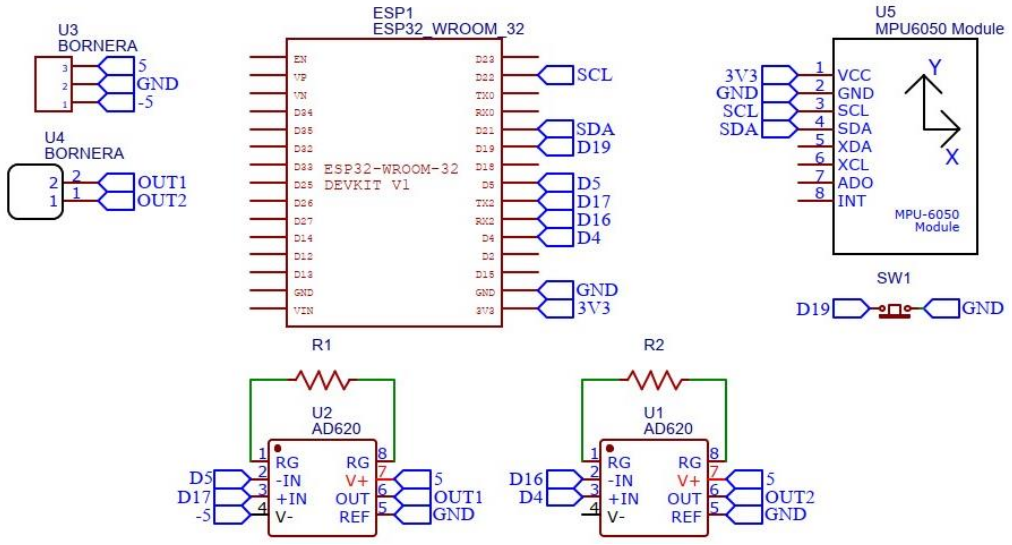
3.7 Documentación, Ajustes y Evaluación

3.7.1 Circuito para generación de pulsos

En esta etapa se realiza la evaluación del sistema, durante la cual se modifica el circuito de recolección de datos incorporando dos amplificadores de instrumentación (uno por cada secuencia de pulsos). El propósito de esta modificación es convertir los pulsos generados por la ESP32 en una señal bifásica. Además, esta etapa tiene el objetivo de emular una neuroprótesis (FES), lo que permite evaluar la eficacia del sistema en condiciones similares a las de una aplicación real de FES. En el siguiente esquema se visualiza el esquemático propuesto.

Figura 16.

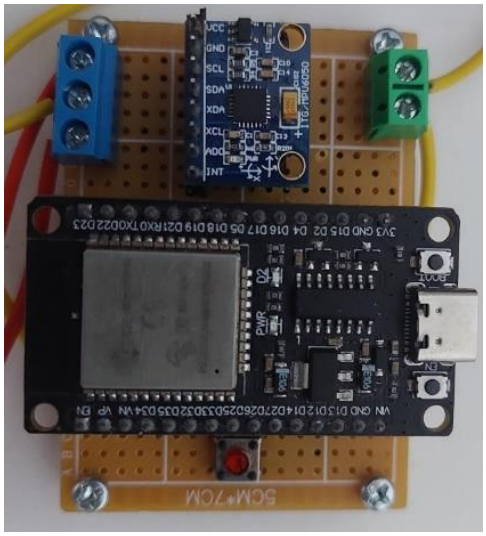
Esquemático para la generación de pulsos FES



Nota. Esquema de conexión del circuito diseñado para generar pulsos bifásicos, simulando el funcionamiento de los pulsos en una neuroprótesis. Recursos propios

Figura 17.

Circuito para la generación de pulsos FES



Nota. Circuito diseñado para generar pulsos bifásicos, simulando el funcionamiento de los pulsos en una neuroprótesis. Recursos propios

3.7.2 Aplicación de técnicas de programación para la reducción del consumo

Existen diversas técnicas para reducir el consumo de energía en la ESP32, como el modo de suspensión profunda “Deep Sleep”, el modo de baja potencia “Light Sleep”, la optimización de la frecuencia del reloj, y la desactivación de periféricos no utilizados. Seleccionamos la última opción, desactivando los periféricos de Wi-Fi y Bluetooth, ya que nuestra aplicación requiere que el dispositivo esté siempre encendido y operativo. Las otras técnicas, aunque efectivas, no eran adecuadas porque interrumpirían el funcionamiento continuo de nuestro sistema.

Se realizaron mediciones de corriente RMS en dos escenarios: en el primero, con los periféricos apagados, en el segundo, con los periféricos sin desactivar.

Tabla 6.

Pruebas de funcionamiento

Consumo con técnica	**Consumo sin técnica**	**Reducción**	**Porcentaje de reducción**
80 mA	85 mA	5 mA	5.88%

Nota. Esta tabla muestra el consumo de la ESP32 en dos diferentes casos.

3.7.3 Pruebas de funcionamiento

Para evaluar el sistema, se realizan pruebas con tres sujetos que ejecutan una serie de movimientos descritos en la tabla que se presenta a continuación:

Tabla 7.

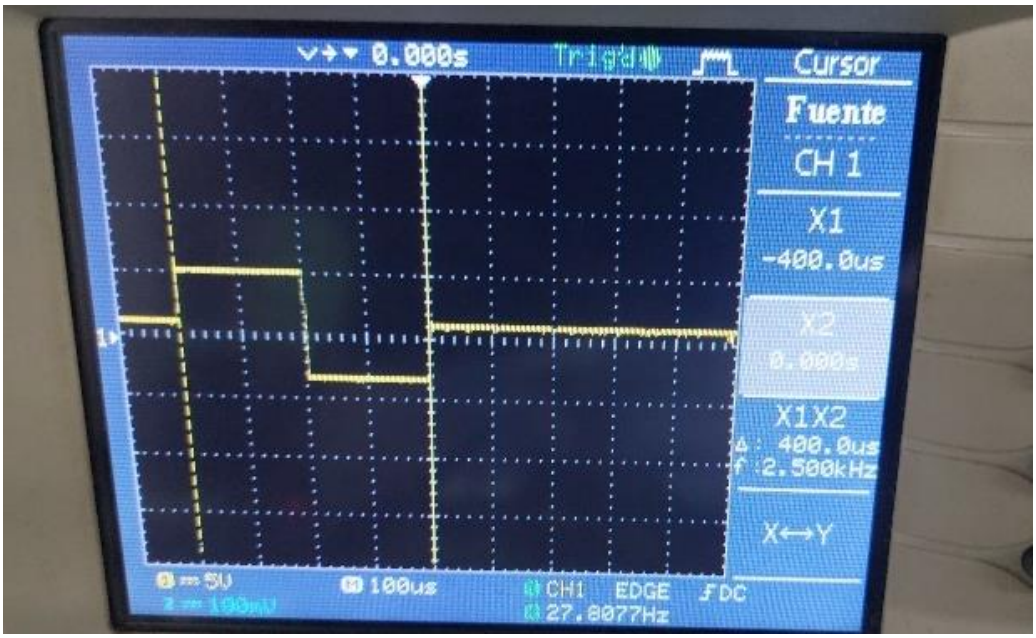
Pruebas de funcionamiento

Sujeto	Brazo	N° pruebas por clase	Movimiento real	Aciertos
Sujeto 1	Izquierdo	10	Apertura	10
			Cierre	9
			Sin acción	10
	Derecho	10	Apertura	9
			Cierre	9
			Sin acción	10
Sujeto 2	Izquierdo	10	Apertura	10
			Cierre	10
			Sin acción	9
	Derecho	10	Apertura	9
			Cierre	8
			Sin acción	10
Sujeto 3	Izquierdo	10	Apertura	9
			Cierre	10
			Sin acción	9
	Derecho	10	Apertura	10
			Cierre	10
			Sin acción	9

Nota. Esta tabla muestra los resultados.

Figura 18.

Pulsos FES generados por el circuito



Nota. Forma de onda en osciloscopio, mostrando el ancho de pulso y la frecuencia. Recursos propios

Los resultados de la validación en pruebas reales indican una precisión del 94%, con 170 movimientos correctamente clasificados de un total de 180.

La clase mejor clasificada en general es "Apertura" y "Sin acción", ambas con un promedio de aciertos de aproximadamente 95%. En contraste, "Cierre" es la clase de menor clasificación con un promedio de aciertos de aproximadamente 93%.

El algoritmo clasifica mejor los movimientos del brazo izquierdo, con un promedio de aciertos de aproximadamente 94%, en comparación con el brazo derecho, que tiene un promedio de aciertos de aproximadamente 93%.

La sensibilidad del reconocimiento del movimiento resultó adecuada para los tres sujetos de prueba, del total de pruebas se generaron 4 detecciones de movimiento erróneas, para tener un resultado cualitativo de la sensibilidad del sistema se realizó una encuesta con los sujetos de prueba.

Figura 19.

Sistema final usado por sujeto

Nota. Implementación final del sistema en un sujeto. Recursos propios

3.7.4 Prueba de tiempo de respuesta

Para evaluar el tiempo de respuesta del algoritmo, se registró el tiempo de ejecución de la función de la red neuronal utilizando la función “micros()” para capturar el tiempo en microsegundos. Se inició el conteo antes de ejecutar la función y se detuvo inmediatamente

después de su finalización. La diferencia entre estos dos tiempos proporcionó el tiempo de procesamiento. Se realizaron 20 movimientos para obtener un promedio de este tiempo.

Figura 20.

Tiempo de respuesta

```
Tiempo de procesamiento de la red neuronal: 175 us
Cambiando a la primera secuencia de pulsos...
Tiempo de procesamiento de la red neuronal: 175 us
Cambiando a la primera secuencia de pulsos...
Tiempo de procesamiento de la red neuronal: 167 us
Tiempo de procesamiento de la red neuronal: 170 us
Cambiando a la primera secuencia de pulsos...
Tiempo de procesamiento de la red neuronal: 169 us
Tiempo de procesamiento de la red neuronal: 174 us
Cambiando a la primera secuencia de pulsos...
Tiempo de procesamiento de la red neuronal: 174 us
Cambiando a la primera secuencia de pulsos...
Tiempo de procesamiento de la red neuronal: 177 us
Cambiando a la primera secuencia de pulsos...
```

Nota. Tiempo de ejecución de la red neuronal. Recursos propios

El tiempo promedio de procesamiento de la red neuronal fue de 170.3 us.

3.7.5 Creación del repositorio

El repositorio del proyecto fue alojado en GitHub para asegurar una distribución accesible del código fuente desarrollado, permitiendo revisar, utilizar y contribuir al proyecto de forma remota. GitHub fue elegido por su capacidad para gestionar versiones del código y su soporte a la colaboración en proyectos de código abierto, lo que facilita el trabajo en equipo, la integración continua y el seguimiento de cambios. Además, el repositorio actúa como respaldo, protegiendo el proyecto ante posibles pérdidas de datos o daños en el sistema local.

El repositorio se organizó de manera modular, con ramas definidas: extracción de características, entrenamiento del modelo, inferencia y dataset, lo que facilita la navegabilidad y comprensión del proyecto para nuevos usuarios o futuros desarrolladores. Cada sección incluye archivos README detallados que explican el uso de los scripts, la instalación de dependencias y la ejecución del proceso completo, lo que contribuye a una documentación clara y accesible.

Se decidió redactar los archivos README y los comentarios en inglés, dado que este es el idioma predominante en la comunidad global de desarrolladores, asegurando que el proyecto pueda ser comprendido por una audiencia internacional. Sin embargo, los nombres de las variables y los mensajes impresos durante la ejecución se mantuvieron en español, en consideración al contexto local y a los usuarios directos que interactuarán con la neuroprótesis.

4. Conclusiones

Se desarrolló un algoritmo que interpreta señales inerciales para identificar patrones de movimiento del brazo, facilitando la apertura y cierre de la mano mediante un emulador de neuroprótesis de Estimulación Eléctrica Funcional. La base de datos se organizó en tres conjuntos: uno para movimientos que indican apertura de la mano (asociados con estirar el brazo), otro para cierre (activado por un movimiento horizontal del brazo) y un tercero para movimientos no relacionados (como la elevación y giro del brazo), lo que ayuda a evitar falsos positivos. La validación en pruebas reales mostró una precisión del 94%, con 170 de 180 movimientos clasificados correctamente. Para optimizar el rendimiento, se utilizaron el optimizador ADAM, la función de activación “ReLU” y regularizadores L1 y L2, además de callbacks de “early stopping” para prevenir el sobreentrenamiento y “ReduceLROnPlateau” para ajustar el “learning rate” del

modelo. El rendimiento del modelo propuesto fue superior al del modelo base, evidenciado por una menor pérdida en el conjunto de entrenamiento.

Se creó un repositorio en GitHub que documenta completamente el algoritmo, detallando la funcionalidad de cada componente y los procedimientos de implementación, lo que permite una distribución accesible del código fuente y facilita la revisión y contribución remota. Este repositorio, organizado en ramas para extracción de características, entrenamiento del modelo, inferencia y “dataset”, mejora la navegabilidad para nuevos usuarios y desarrolladores. Cada sección incluye archivos README en inglés, que explican el uso de los scripts y la instalación de dependencias, mientras que los nombres de las variables y mensajes impresos se mantienen en español para los usuarios locales. Esta documentación actúa como una guía para realizar mejoras y ajustes en el algoritmo, optimizando la extracción de características y la arquitectura de la red neuronal.

Las pruebas realizadas con un circuito emulador de la neuroprótesis FES demostraron que el algoritmo ofrece un alto rendimiento en precisión, sensibilidad y tiempo de respuesta. Los resultados de la validación en pruebas reales indicaron una precisión del 94%, clasificando correctamente 170 de 180 movimientos. La clase mejor clasificada fue "Apertura" y "Sin acción", ambas con un promedio de aciertos de aproximadamente 95%, mientras que "Cierre" tuvo una precisión menor, con un promedio de 93%. El algoritmo mostró un mejor desempeño en los movimientos del brazo izquierdo (94%) en comparación con el derecho (93%). La sensibilidad del reconocimiento fue adecuada para los tres sujetos de prueba, con solo cuatro detecciones erróneas, y una encuesta reveló que el 66.7% de los participantes consideraron que los movimientos fueron bien reconocidos. Además, el tiempo promedio de procesamiento de la red neuronal fue de 170.3

microsegundos. Los usuarios afirmaron que el LED indicaba la acción en el momento adecuado, aunque un 33.3% mencionó que, en ocasiones, se encendía con movimientos muy leves.

La gestión del estado de los periféricos en el ESP32 evidenció una reducción en el consumo de energía al desactivar los módulos de Bluetooth y Wi-Fi. Esta técnica fue seleccionada porque la aplicación requiere que el dispositivo esté siempre encendido y operativo, y otras técnicas de reducción de energía no eran adecuadas debido a su impacto en el funcionamiento continuo del sistema. Las mediciones de corriente RMS realizadas en dos escenarios mostraron un consumo de 80 mA con los periféricos apagados, en comparación con 85 mA cuando estaban activos, resultando en una reducción de 5 mA, equivalente al 5.88%.

El algoritmo desarrollado buscó un equilibrio entre eficacia y uso adecuado de los recursos, considerando las limitaciones del microcontrolador utilizado y la necesidad de obtener una respuesta rápida en tiempo real. Dado que el procesamiento de la red neuronal debía ser realizado directamente por el microcontrolador, se optó por una arquitectura no demasiado profunda, lo que permitió mantener un rendimiento adecuado sin comprometer la velocidad de respuesta ni sobrecargar los recursos del sistema.

5. Recomendaciones

Incorporar características adicionales del sensor en el entrenamiento de la red neuronal podría mejorar el desempeño del sistema. Ampliar la variedad de datos utilizados para entrenar el algoritmo puede ayudar a capturar mejor las variaciones en los movimientos y optimizar la precisión en la clasificación de intenciones de movimiento.

Para validar el desempeño del algoritmo en condiciones más variadas y reales, se recomienda realizar pruebas adicionales en diferentes escenarios y con una mayor diversidad de usuarios. Esto ayudará a evaluar cómo el sistema se comporta en situaciones del mundo real y a identificar posibles áreas de mejora.

Probar el algoritmo en una neuroprótesis funcional, en lugar de en un circuito emulador, permitirá evaluar su desempeño en condiciones de uso auténticas. Esta implementación proporcionará una visión más precisa sobre su efectividad y ajuste en situaciones prácticas, lo que puede revelar aspectos importantes sobre su rendimiento y potencial de mejora.

Se recomienda diseñar todo el sistema físico, incluyendo tanto la neuroprótesis como el microcontrolador, en una sola PCB (Printed Circuit Board) personalizada. Esto permitirá un mayor control sobre la arquitectura del hardware, aprovechando el potencial completo del microcontrolador y eliminando limitaciones inherentes a módulos preconstruidos. Al desarrollar una PCB específica para este propósito, se maximizará la eficiencia del dispositivo y se reducirá el consumo energético al eliminar componentes innecesarios.

Es fundamental obtener los permisos éticos necesarios para trabajar con personas que presenten discapacidades físicas, lo que permitirá recolectar datos de calidad basados en sus necesidades específicas. Con estas bases de datos, será posible entrenar los algoritmos de la neuroprótesis considerando las dificultades particulares de cada usuario. Esto asegurará que el dispositivo esté optimizado para los movimientos que cada individuo necesita realizar, mejorando su eficacia en el uso real.

En lugar de crear un algoritmo genérico que requiera que los usuarios se adapten a él, se recomienda desarrollar una base de datos personalizada para cada individuo. Esto permitiría que el algoritmo de control de la neuroprótesis se ajuste al 100% a las características únicas de cada

persona, logrando una personalización total del sistema. De esta forma, la prótesis ofrecería una experiencia más natural y eficiente, respondiendo a las necesidades individuales sin comprometer la comodidad o la funcionalidad.

Referencias Bibliográficas

Amer Cuenca, J. J. (2011). Efectos analgésicos de la aplicación de Estimulación Nerviosa Eléctrica Transcutánea en colonoscopias sin sedación. 40.

Aprendizaje Automático para Arduino, E. y. (2019, Septiembre). *Udemy*. Retrieved from https://www.udemy.com/course/draft/2518870/

Aramayo, A. (2017, Mayo 7). *Terminología básica en electroterapia - Phisiobasic. Phisiobasic.* . Retrieved from https://phisiobasic.com/terminologia-basica-en-electroterapia/

Arias, Á. (2009). Rehabilitación del ACV: evaluación, pronóstico y tratamiento. *Galicia Clínica*, 25-40.

Buergo Zuaznábar, M. Á., Fernández, O., Pérez, J., & Pando, A. (2007). Guías de práctica clínica para la prevención primaria, el manejo en la fase aguda y la prevención secundaria del ictus. *Enfermedades cerebrovasculares*, 3.

Calderón, R. F. (2019). Desarrollo de un sistema portátil de estimulación eléctrica funcional (FES) para pacientes con síndrome de pie caído. 25-28.

Delis, A. F. (2013). Toward an upper limb motor neuroprosthesis for neurorehabilitation. *Pan American Health Care Exchanges (PAHCE)*, 1.

El Naqa, I. &. (2015). *What is machine learning?* Springer International Publishing.

Fernandez, P. (2021, Diciembre 6). *Medium*. Retrieved from The Difference between Artificial Intelligence, Machine Learning, and Deep Learning: https://medium.com/@peter.fernandez/the-difference-between-artificial-intelligence-machine-learning-and-deep-learning-4cf71a5a2

Fisioterapia, M. (2020, Marzo 20). *Introducción a la electroterapia: Dispositivos, ondas y parámetros de la corriente eléctrica.* Retrieved from mirandafisioterapia: https://www.mirandafisioterapia.com/post/introduccion-a-la-electroterapia-dispositivos-ondas-y-parametros

Géron, A. (2022). *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow* (Vol. II). O'Reilly Media, Inc.

Lin, H. -Y. (2023, Septiembre). Embedded Artificial Intelligence: Intelligence on Devices. *Computer, 56*(9), 90-93.

Lynch, C. L. (2008). Functional electrical stimulation. *IEEE control systems magazine, 28*(2), 40-50.

Mascret, Q. G.-T. (2022). A wearable sensor network with embedded machine learning for real-time motion analysis and complex posture detection. *IEEE Sensors Journal, XXII*(8), 7868–7876.

Ramos, O. L. (2016). Reconocimiento de patrones vocálicos mediante la implementación de una red neuronal artificial utilizando sistemas embebidos. *Información tecnológica*, 133-142.

Rehabtronics. (s.f.). *ReGrasp - Rehabtronics*. Obtenido de https://rehabtronics.com/product/regrasp-rehabilitation-glove/

Rouhiainen, L. (2018). *Inteligencia artificial.* Madrid: Alienta Editorial.

Samuel, A. (1959). Aprendizaje automático. *The Technology Review*, pp. 42-45.

Shi, Z. Y. (2020). Artificial intelligence techniques for stability analysis and control in smart grids: Methodologies, applications, challenges and future directions. *Applied Energy*, 278.

Thrasher, M. P. (2004). "Neuroprótesis" . *Enciclopedia de biomateriales e ingeniería biomédica, II*, 1056-1065.

Tresadern, P., Thies, S., Kenney, L., Howard, D., & Goulermas, J. Y. (2006). Artificial neural network prediction using accelerometers to control upper limb FES during reaching and grasping following stroke. *International Conference of the IEEE Engineering in Medicine and Biology Society*, 2916-2919.

Vega, L. C. (2019). Revisión sobre la efectividad de la terapia en espejo en el proceso de rehabilitación de miembros superiores en pacientes con accidente cerebrovascular. *Movimiento científico*, 47-54.

Williamson, & Andrews. (2000). Gait event detection for FES using accelerometers and supervised machine learning. *Transactions onRehabilitation Engineering*, 312-319.

Zhang, Z. &. (2023). A review of artificial intelligence in embedded systems. *Micromachines, XIV*(5), 897.

Apéndices

Apéndice A. Algoritmo de IA para Neuroprótesis con Sensor MPU6050

Este repositorio contiene el código fuente completo y comentado de un algoritmo de inteligencia artificial desarrollado para controlar una neuroprótesis utilizando datos del sensor MPU6050. Los archivos están organizados por secciones clave para facilitar su comprensión y uso:

- **Extracción de características:** Scripts encargados de procesar los datos del sensor y extraer las características relevantes.
- **Creación y entrenamiento del modelo:** Código para definir, entrenar y evaluar el modelo de IA.
- **Inferencia:** Scripts para realizar predicciones usando el modelo ya entrenado.
- **"Dataset":** El conjunto de datos utilizado para entrenar el modelo está incluido para pruebas y experimentación.
- **Instrucciones:** Archivos README con guías detalladas sobre cómo usar los scripts, instalar dependencias y ejecutar el proceso completo.

Todo el código está completamente comentado para facilitar su comprensión y cada sección cuenta con su respectivo README con instrucciones claras sobre cómo ejecutar los scripts.

Puede acceder al repositorio en el siguiente enlace: Neuroprosthesis Control via AI and MPU6050 Sensor.

Apéndice B. Resultados de la Encuesta de Satisfacción del Sistema de Generación de Pulsos mediante IA

Esta encuesta se realizó para evaluar la satisfacción de los usuarios con el sistema de generación de pulsos basado en un algoritmo de inteligencia artificial. El formulario fue diseñado para recopilar opiniones de los participantes que probaron el sistema, con el objetivo principal de evaluar la sensibilidad del sistema al generar los pulsos.

Se plantearon tres preguntas cualitativas a los tres sujetos que participaron en las pruebas del algoritmo:

¿El LED se encendió en el momento correcto al realizar la acción?

¿Qué tan bien cree que el sistema reconoció su acción?

¿El algoritmo detectó movimiento cuando no se realizó la acción?

Cada pregunta buscaba medir la precisión y fiabilidad del sistema desde la perspectiva de los usuarios. Estos resultados proporcionarán valiosa información para mejorar la sensibilidad y eficiencia del sistema de detección.

Puede acceder al documento completo con los resultados de la encuesta en el siguiente enlace: Resultados de la Encuesta.

Apéndice C. Curso de Udemy: Desarrollo de Algoritmos de IA para el Control de Neuroprótesis

Este proyecto se basó en gran medida en el contenido del curso de Udemy, donde se proporcionan los conocimientos y archivos clave que fueron utilizados en el desarrollo de nuestro sistema de control de neuroprótesis mediante inteligencia artificial y el sensor MPU6050. Los materiales y documentos incluidos en el curso sirvieron como base para la implementación de los algoritmos y técnicas de procesamiento de datos que empleamos.

El curso proporciona una guía detallada para el desarrollo de soluciones prácticas con IA, lo que facilitó el diseño de las etapas de extracción de características, entrenamiento del modelo y pruebas de inferencia en nuestro proyecto.

Puede acceder al curso a través del siguiente enlace: Udemy Course.

Apéndice D. Video de funcionamiento

En este video se presenta una prueba del funcionamiento del algoritmo, junto con el circuito emulador de la neuroprótesis FES.

Puede acceder al video a través del siguiente enlace: Video de funcionamiento