SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

Implementación de algoritmo de machine learning en un sistema embebido para control de prótesis activa utilizando señales mioeléctricas

PROYECTO INTEGRADOR

Previo la obtención del Título de:

Ingeniería en Electricidad especialización Electrónica y Automatización Industrial

Presentado por:
Galo Sánchez Granja

GUAYAQUIL - ECUADOR
Año: 2021

DEDICATORIA

A mis padres por todo su apoyo incondicional y por siempre motivarme a alcanzar mis metas.

DECLARACIÓN EXPRESA

“Los derechos de titularidad y explotación, me corresponde conforme al reglamento de propiedad intelectual de la institución; Galo Sánchez Granja y doy mi consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

Galo Sanchez Granja

EVALUADORES

VICTOR MANUEL ASANZA ARMIJOS

Firmado digitalmente por VICTOR MANUEL ASANZA ARMIJOS
Fecha: 2021.05.03 19:12:41 -05'00'

PhD. Wilton Agila
PROFESOR DE LA MATERIA

Msc. Victor Asanza
PROFESOR TUTOR

RESUMEN

El campo del desarrollo de prótesis activas es un campo en constante evolución y crecimiento, en la actualidad encontramos prótesis cada vez más sofisticadas las cuales pueden emular los movimientos complejos de su contraparte biológica. Con el incremento en la complejidad de los movimientos que puede realizar una prótesis activa, sale a la luz el desafió que existe al momento de diseñar los controladores para prótesis, los cuales no solo deben controlar el movimiento de la prótesis, sino también interpretar de forma precisa la intención de movimiento del usuario. Para este trabajo se planteó, utilizando machine learning y sistemas embebidos, diseñar un sistema de control de prótesis activa basado en la interpretación de la intención motora usando electromiografía. Para cumplir con los objetivos planteados se creó un conjunto de datos de entrenamiento con los casos de interés a ser clasificados por el algoritmo de machine learning, luego basado en estos datos, se entrenó una red neuronal. Este algoritmo fue evaluado en distintas plataformas para establecer cuál de ellas ofrece el mejor rendimiento. Como resultado se obtuvo un sistema de clasificación cuya precisión dentro del conjunto de datos de entrenamiento es del 100%, además de las distintas plataformas evaluadas se encontró que el Arduino Nano ofrece la mejor relación entre desempeño y rendimiento. En conclusión el uso de plataformas de bajo costo, y algoritmos sofisticados de clasificación permite el diseño de un sistema que es robusto, funcional y de bajo costo.

Palabras Clave: Sistemas embebidos, Inteligencia Artificial, Electromiografía, Prótesis activa.

ABSTRACT

The field of active prosthesis development is a field in constant evolution and growth, nowadays we find increasingly sophisticated prostheses which can emulate the complex movements of their biological counterpart. With the increase in the complexity of the movements that an active prosthesis can perform, the challenge that exists when designing the prosthesis controllers comes to light, which must not only control the movement of the prosthesis, but also interpret accurately the user's movement intention. For this work, it was proposed, using machine learning and embedded systems, to design an active prosthesis control system based on the interpretation of motor intention using electromyography. To meet the proposed objectives, a training data set was created with the cases of interest to be classified by the machine learning algorithm, then based on these data, a neural network was trained. This algorithm was evaluated on different platforms to establish which of them offers the best performance. As a result, a classification system was obtained whose precision within the data set is 100%, in addition to the different platforms evaluated, it was found that the Arduino Nano offers the best relationship between performance and performance. In conclusion, the use of low-cost platforms and sophisticated classification algorithms allows the design of a system that is robust, functional, and low-cost.

Keywords: embedded systems , Artificial Intelligence, electromyography, active prosthetics.

ÍNDICE GENERAL

ABREVIATURAS

ESPOL	Escuela Superior Politécnica del Litoral
EMG	Electromiografía
EGG	Electroencefalograma
AI	Artificial Intelligence
GUI	Graphical user interface
ADC	Analog to Digital Converter
RMS	Root Mean Square

SIMBOLOGÍA

mV	milivoltios
uV	nanovoltios

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

1.Introducción

1.1 Descripción del problema

El campo del desarrollo de prótesis activas es un campo en constante evolución y crecimiento, en la actualidad encontramos prótesis cada vez más sofisticadas las cuales pueden emular los movimientos complejos de su contraparte biológica. Con el incremento en la complejidad de los movimientos que puede realizar una prótesis activa, sale a la luz el desafió que existe al momento de diseñar los controladores para prótesis, los cuales no solo deben controlar el movimiento de la prótesis, sino también interpretar de forma precisa la intención de movimiento del usuario [1].

Como interfaz entre el usuario y la prótesis, se usa comúnmente señales biológicas generadas por el usuario siendo las más populares las señales de electroencefalografía (EEG), y las señales de electromiografía (EMG). De estas dos señales la EEG ha resaltado por generar sistemas con la capacidad de interpretar con mayor precisión la intención motora del usuario, debido a que esta señal nos permite conocer y caracterizar la actividad cerebral [2]. Sin embargo existen inconvenientes inherentes cuando se realizan diseños basados en EEG; la actividad cerebral genera patrones de voltaje muy pequeños, en el orden de los micro voltios, además estas señales son más susceptibles al ruido electromagnético que las señales de EMG, sin mencionar que para realizar la adquisición de datos se deben colocar varios electrodos en la superficie craneal del usuario. Por estos motivos el uso de EMG se ha convertido en el estándar de facto para prótesis activas [3].

Actualmente vivimos en un era donde la inteligencia artificial está presente en la mayoría de los aspectos de nuestra vida [4]. Machine Learning es el campo de estudio que da a las computadoras la capacidad de realizar aprendizaje basado en datos para realizar una acción sin ser programadas de forma explícita para ello [5]. Esta rama de las ciencias computacionales ha encontrado su espacio en el sectores financieros, de entretenimiento digital, ingeniería, medicina, entre otros, donde destaca principalmente por su precisión y su facilidad de implementación. Los

algoritmos de Machine Learning presentan un desafió al momento de ser implementados, estos consumen bastantes recursos computacionales debido a la complejidad de las operaciones realizadas, por esto comúnmente se alojan en servidores donde la capacidad de procesamiento es mayor [6].

Los microcontroladores son ampliamente utilizados en la industria y la academia, debido a su flexibilidad y bajo costo, sin embargo estos factores causantes de su popularidad son también su limitante [7]. Comúnmente los procesadores dentro de los microcontroladores más populares son procesadores de 8 bits los cuales pueden manejar un máximo de memoria de 65536 bytes. La combinación de escasa memoria disponible y el limitado número de instrucciones por segundo que un procesador de bajo costo puede realizar, traen consigo un desafío para los diseñadores de controladores de prótesis activas.

1.2 Justificación

La implementación de algoritmos de machine learning en sistemas embebidos puede dar como resultado productos sofisticados de bajo costo, los cuales pueden ser reproducidos a gran escala para así proveer de una solución económica y de mayor alcance en el mercado. Para esto se debe realizar un estudio que determine el tipo de algoritmo a utilizar y el sistema embebido donde se aloja este programa, de tal forma que el precio y el desempeño puedan ser priorizados al momento de desarrollar una solución comercial al problema del control de prótesis activas.

1.3 Objetivos

1.3.1 Objetivos generales

Diseñar e implementar un algoritmo de machine learning capaz de interpretar la intención motora del usuario, el cual deberá ser ejecutado en un sistema embebido de bajo costo, para así poder controlar una mano prostética.

1.3.2 Objetivos específicos

- Crear un conjunto de datos lo suficientemente grande, capaz de representar los casos de uso de la prótesis, y así entrenar el modelo de inteligencia artificial para poder interpretar la intención motora del usuario
- Traducir el modelo de inteligencia artificial entrenado de un lenguaje de alto nivel, a un lenguaje de bajo nivel, de tal forma que el código utilice de mejor forma los recursos computacionales del sistema embebido.
- Probar el modelo de interpretación de intención motora optimizado en diferentes sistemas embebidos de bajo costo, establecer los parámetros de uso de memoria y rendimiento, y establecer cuál plataforma ofrece la mejor relación entre costo y desempeño.

1.4 Marco teórico

1.4.1 Control basado en EMG

La interfaz de control basado en electromiografía es el método más común usado para el control de prótesis activas[8]. Las prótesis mioeléctricas son controladas por medio de la medición de señales electromiográficas provenientes de dos músculos residuales independientes o por medio de la distinción de diferentes niveles de acción de un músculo residual. Avances recientes en algoritmos de reconocimiento de patrones y procedimientos quirúrgicos como la reinvención muscular dirigida están siendo desarrollados con el objetivo de mejorar el funcionamiento de la interfaz basada en EMG [9-11].

Las señales electromiográficas de superficie, las cuales son recolectadas con electrodos situados en la superficie de la piel sobre el músculo, reflejan sintéticamente una reacción eléctrica del potencial accionar generado por las unidades motoras. Estas señales biomédicas contienen información rica sobre la intención motora del usuario y por lo tanto pueden ser utilizadas para el control de dispositivos de rehabilitación y de asistencia robótica, tales como las manos protésicas o exoesqueletos.

Con el objetivo de mejorar la calidad de vida de las personas amputadas, cada nueva iteración de mano prostética diseñada aumenta la cantidad de grados de libertad de

la prótesis, para así poder cubrir las actividades diarias del amputado. Sin embargo, solo unas cuantas de estas actividades pueden ser realizadas utilizando controles tradicionales basados en electromiografía. Para lograr un control intuitivo de la mano prostética de múltiples grados de libertad, los métodos populares de control basado en EMG, tienden a utilizar tecnología basada en reconocimiento de patrones, el cual está compuesto de dos procesos, extracción de características y clasificación de patrones.

Existen dos clases principales de contracción esqueleto muscular: contracción isotónica y contracción isométrica. La contracción isotónica comprende cambios en la longitud del músculo, y está siempre relacionada a la dinámica del movimiento del cuerpo; mientras que la contracción isométrica no involucra ningún cambio en la longitud del músculo, y está ampliamente presente en posturas musculares estáticas.

1.4.2 Machine Learning

Machine Learning es una rama de la ciencias computacionales, la cual se enfoca en programar computadoras de tal forma que puedan realizar aprendizaje basado en datos, y poder así realizar tareas para las cuales no fueron programados de forma explícita.

El filtro de spam en nuestro sistema de correo electrónico es un ejemplo de programa basado en Machine Learning que, dado un conjunto de correos spam y correos no spam, puede aprender a identificar entre un tipo de correo u otro [5].

Otra de las tareas que pueden realizar los algoritmos de Machine Learning es la predicción del comportamiento de un sistema basado en su historia. Existen muchos tipos de predicción que pueden ser realizadas. Podemos predecir el resultado de un evento que ocurre en el futuro en situaciones como el mercado financiero, el clima el día mañana, la siguiente palabra o texto que será usado en un mensaje, o anticipar el comportamiento de un peatón dentro de un sistema de manejo automático de un vehículo, entre otras.

1.4.2.1 Etiquetas y Conjunto de datos

Para realizar la optimización del modelo de inteligencia artificial, se necesita un conjunto de datos relacionados a los datos al cual el modelo estará expuesto una vez entrenado. Los datos de entrenamiento son la entrada del modelo, el valor de los datos de salida dentro del conjunto de datos de entrenamiento se conoce como etiqueta, de esta forma el modelo es optimizado para producir la misma salida dado un conjunto de datos de entrada.

1.4.2.2 Tipos de Machine Learning

Existen muchos tipos diferentes de sistemas de Machine Learning que sería útil clasificarlos en categorías amplias, dependiendo de los siguientes criterios:

- Si existe o no supervisión humana al momento de realizar el entrenamiento del modelo de inteligencia artificial (Supervised, Unsupervised, Semisupervised, y Reinforcement Learning)
- Si puede o no realizar aprendizaje desde cero durante la ejecución
- Si trabajan simplemente comparando datos de entradas nuevos, con puntos conocidos, o en cambio detectando patrones en el conjunto de datos de entrenamiento y construyendo un modelo (KNN).

Supervised Learning, o aprendizaje supervisado, es el tipo de aprendizaje de máquina más estudiado dentro del campo de la inteligencia artificial. Se basa principalmente en construir de forma automática una función que relaciona entradas, con salidas, basado en el par entrada-salida dentro de los datos de entrenamiento.

En el otro extremo, existe el aprendizaje sin supervisión o Unsupervised Learning, el cual asume total ausencia de etiquetas, es decir que dentro del conjunto de datos no existe el par entrada-salida, simplemente un conjunto amplio de entradas. El objetivo principal es aprender la estructura escondida dentro del conjunto de datos disponibles.

Reinforcement Learning se encuentre en medio del aprendizaje supervisado y el aprendizaje sin supervisión. Por un lado, utiliza muchos métodos bien establecidos del aprendizaje supervisado, redes neuronales para aproximación de funciones por ejemplo. Por otro lado, se aplica de forma distinta, debido a que no existen etiquetas establecidas, en cambio, el aprendizaje se maneja a través de recompensas alcanzadas al realizar acciones dentro de un ambiente determinado [12].

1.4.3 Sistemas Embebidos

Un sistema embebido es una computadora basada en un microprocesador con software dedicado a realizar una función específica, ya sea como un sistema independiente, o como parte de un sistema compuesto. En su núcleo existe un circuito integrado diseñado para llevar a cabo operaciones computacionales en tiempo real [13].

Desde el punto de vista de la programación los sistemas embebidos son diferentes en muchas formas a otras plataformas de software. El hardware donde los programas son ejecutados está frecuentemente restringido en terminas de cantidad de memoria disponible y la frecuencia del procesador. Sin embargo, los sistemas embebidos controlan frenos de carros, señaléticas de tránsito, equipo médico y otros dispositivos críticos para la vida.

1.4.3.1 Arduino UNO

El Arduino Uno es una tarjeta microcontrolador basado en el ATmega328P. Tiene 14 entradas/salidas digitales (de las cuales 6 pueden ser utilizadas como salidas PWM), 6 entradas analogicas, un resonador cerámico de 16MHz, conexión usb, entre otros componentes necesarios para el funcionamiento del microcontrolador [14].

Tabla 1.1 Ficha técnica del microcontrolador Arduino Uno

Microcontrolador	ATmega328P
Voltaje de operación	5 V
Voltaje de entrada	7 - 12 V
E/S Digital	14
Salidas PWM	6
Frecuencia de reloj	16 MHz
Memoria flash	32 kb de los cuales 0.5 Kb usado para el bootloader
SRAM	2 Kb
EEPROM	1 Kb

1.4.3.2 Arduino Nano Every

El Arduino Nano Every es la tarjeta más pequeña disponible en la familia Arduino, preferida principalmente para proyectos donde se requiere el uso de microcontroladores pequeños y de fácil uso. La combinación entre bajo costo y tamaño (45x18mm) convierte a este dispositivo en el preferido para aplicaciones vestibles, robótica de bajo costo, entre otras [15].

Tabla 1.2 Ficha técnica del microcontrolador Arduino Nano Every

Microcontrolador	ATmega4809
Voltaje de operación	5 V
E/S Digital	14
Salidas PWM	5
Frecuencia de reloj	20 MHz
Memoria flash	48 KB
SRAM	6 KB
EEPROM	256 byte

1.4.3.3 MCU ESP8266

El ESP8266 es un chip WiFi de bajo costo, con soporte completo de TCP/IP, y capacidad de microcontrolador, producido por Espressif Systems en Shanghai China. Este pequeño módulo permite que los microcontroladores se conecten a wifi y realizar una coneccion TCP/IP usando comandos de estilo Hayes. Sin embargo, al principio del lanzamiento de este producto no existía documentación en otro idioma diferente al Chino, por lo que su acogida llegó unos años después [16].

Tabla 1.3 Ficha técnica del microcontrolador ESP8266

Microcontrolador	L106 32-bit RISC
Frecuencia de reloj	80 MHz
Memoria Ram ● Ram de instrucciones ● Ram de caché de instrucciones ● Ram de datos de usuario	 32 KB 32 KB 80 KB
GPIO	16
ADC	10 bit

1.4.4 Gym de OpenAI

Gym es una librería desarrollada por la compañía especializada en inteligencia artificial OpenAI. La librería es una colección de ambientes utilizados principalmente para probar algoritmos de Reinforcement Learning [17], [19], [20].

1.4.4.1 Ambiente HandReach-v0

El ambiente HandReach de la librería Gym simula la mano robótica ShadowHand, el objetivo de este ambiente es diseñar un controlador el cual pueda alcanzar una posición generada de forma aleatoria, mediante el control de los actuadores de la mano. La mano robótica cuenta con 24 grados de libertad [18].

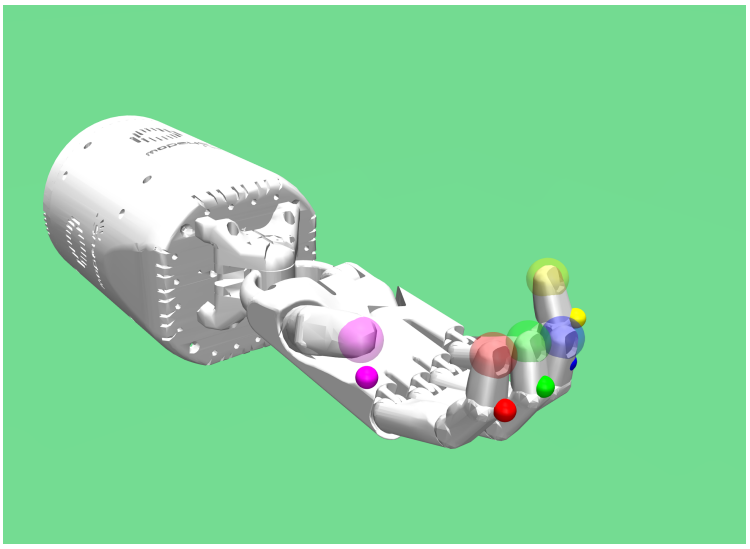

Figura 1.1 Representación multimedia del ambiente de simulación HandReach de la libreria Gym de OpenAi

1.4.5 MyoWare de Advancer Technologies

Este sensor mide, filtra , rectifica y amplifica la actividad eléctrica del músculo; dando como resultado valores que van de 0 a Vs, dependiendo de la intensidad de la actividad muscular realizada, donde Vs es el voltaje de alimentación. Este sensor está diseñado para trabajar con microcontroladores, por lo tanto trabaja con voltajes de alimentación de 3.3 y 5 voltios [21].

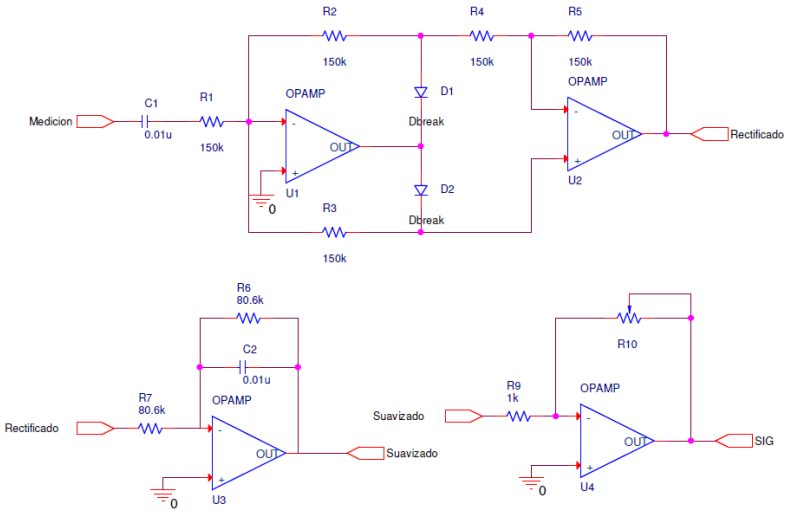

Figura 1.2 Circuito analogico utilizado en el sensor Myoware para el preprocesamiento de señales mioeléctricas

El sensor utiliza únicamente electrónica analógica, diseñado alrededor del dispositivo AD8648 de Texas Instruments, un circuito integrado que contiene 4 amplificadores operacionales, dispuestos en configuración de rectificador, integrador y amplificador, como se observa en la figura 1.2, los cuales son utilizados para el procesamiento de la señales eléctricas adquiridas a través de los electrodos [22]. Se colocan tres electrodos para realizar la adquisición de datos, uno al inicio del área de interés, otro al final, y uno como referencia para el sistema, como se muestra en la figura 1.3.

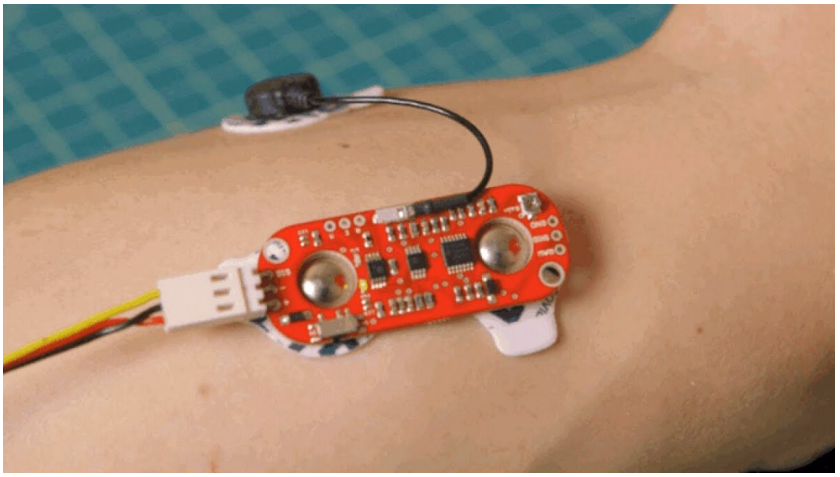

Figura 1.3 Adquisición de señales mioeléctricas de superficie a través de electrodos utilizando el sensor Myoware

CAPÍTULO 2

2. Metodología

El desarrollo de este trabajo se dividió en 3 procesos secuenciales, adquisición de las señales mioeléctricas del antebrazo, entrenamiento del modelo de clasificación de intención motora, y por último la implementación y evaluación del modelo de inteligencia artificial en distintos sistemas embebidos.

2.1 Adquisición de datos

Con el fin de controlar la mano robótica alojada en el ordenador utilizando señales mioeléctricas, se implementó el sistema de adquisición de datos, utilizando el sensor Myoware y el ADC del Arduino Uno. Para realizar una captura de datos adecuada se tuvo en cuenta ciertas recomendaciones realizadas por el fabricante del sensor, los electrodos conectados al amplificador de instrumentación deben estar conectados en medio del músculo de interés y alineado a las fibras musculares, de lo contrario la señal se verá atenuada como se muestra en la figura 2.1, además el electrodo de referencia actúa mejor en áreas con mayor presencia de hueso bajo la piel. Con el propósito de obtener la información de mayor fidelidad a la intención de movimiento, los electrodos se colocaron en la parte interna del antebrazo donde se encuentran los músculos flexores encargados de cerrar la mano.

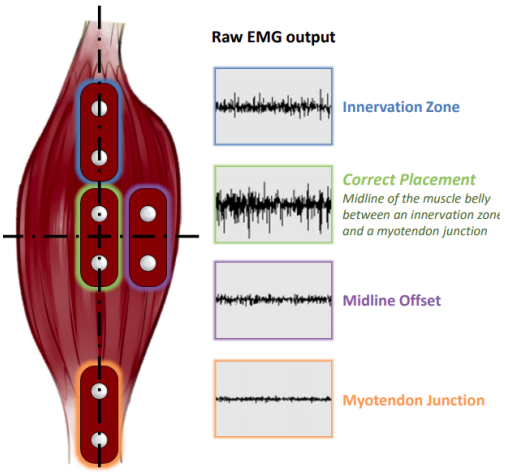

Figura 2.1 Importancia de la correcta ubicación de los electrodos

Otro aspecto que se tuvo en cuenta fue la frecuencia de muestreo al momento de tomar los datos. Según la regla de Nyquist, se debe utilizar una frecuencia de

muestreo al menos 2 veces mayor a la componente de mayor frecuencia dentro de la señal de interés. La frecuencia más rápida en las señales de electromiografía se encuentra dentro del rango de los 400 a 500 Hz, por lo tanto se utilizó como frecuencia de muestreo 1000Hz. Cabe recalcar que dentro del ambiente de programación de Arduino no existen opciones explícitas para ajustar la frecuencia de muestreo, por lo tanto, para realizar la adquisición de datos a la frecuencia deseada se utilizó el código que se muestra en la figura 2.2.

```
void record(){                          //Funcion de adquisicion y envio de datos
  int i;
  for (i=0;i<2000;i++)
  {
    data[i]=analogRead(emg);            //Lectura y almacenamiento del puerto analogico
    delay(1);                           //delay de 1ms
  }
  for (i=0;i<2000;i++)
  {
    Serial.println(data[i]);            //comunicacion serial al ordenador
    delay(1);
  }
}
```

Figura 2.2 Función *record* utilizada para la creación del conjunto de datos de entrenamiento

La función delay pausa el programa por la cantidad de tiempo especificada en milisegundos al momento de utilizar la función, de esta forma se evita la ejecución de la siguiente instrucción hasta cumplir con el tiempo especificado. Se seleccionó una ventana temporal de 2 segundos durante los cuales se deberán tomar 2000 datos de la actividad muscular generada al momento de realizar los movimientos que serán interpretados. Teniendo en cuenta ambos parámetros, el tiempo de duración del evento y la frecuencia de muestreo, se escribió un programa el cual, de forma implícita ajusta la frecuencia de muestreo a la cantidad deseada.

Para almacenar los datos enviados desde el arduino hacia el ordenador, se utilizó la herramienta pyserial, una librería de python que facilita la manipulación de los puertos seriales del ordenador. La figura 2.3 muestra el código utilizado para el almacenamiento de datos, este programa se encarga de almacenar cada nueva entrada como un archivo de texto plano con el nombre de *data_N.txt* , donde N es el número en orden ascendente de entradas almacenadas en el ordenador.

```python
import serial
arduino = serial.Serial('COM9', 9600, timeout=.1)   #Instanciacion y Configuracion del puerto serial
file_name = 'data_'                                 #Nombre del archivo
n_file = 0                                          #Numero de archivo
contador = 0                                        #Contador de datos recibidos
while True:
    data = arduino.readline()[:-2]
    file = file_name +  str(n_file) + '.txt'        #Instancia del archivo
    f = open(file,'a')                              #Se abre el archivo en configuracion append
    if data:                                        #Si se recibe un dato entonces:
        f.write(str(int(data))+'\n')                #Se escribe el dato en el archivo
        contador = contador + 1                     #El contador de datos aumenta
        print(str(data))                            #Se muestra el dato por pantalla

    if contador==2000:                              #Si la cantidad de datos recibidos es mayor a 2000
        f.close()                                   #Se cierra el archivo donde estaba escribiendo
        n_file = n_file + 1                         #El siguiente archivo tendra +1 en numeracion
        contador = 0                                #Se reinicia el contador de datos
```

Figura 2.3 Programa de almacenamiento de datos recibidos a través del puerto serial

Durante la toma de datos se ensayaron 4 casos distintos, dos casos de actividad muscular dinámica, y dos casos de actividad muscular estática. El primer caso estático es cuando el usuario de la prótesis no desea realizar ningun acción, simplemente mantener la mano abierta, el algoritmo debe ser capaz de interpretar este deseo, debido a que, a pesar de la inactividad muscular, los electrodos captan patrones de señales eléctricas de forma continua, los cuales pueden ser interpretados como movimiento sino se consideran dentro del diseño del algoritmo de inteligencia artificial. El segundo caso de actividad muscular estática, es la intención de mantener la mano cerrada. Por último los dos casos de actividad muscular dinámica comprenden la transición entre ambos casos estáticos, es decir, cuando el usuario desea pasar de tener la mano abierta en reposo a tener la mano cerrada, pasa por una transición donde las señales de electromiografía cambian generando un patrón el cual indica el deseo de pasar de un estado a otro. Para cada caso se tomaron 50 muestras con el fin de obtener suficiente información para poder entrenar el modelo de inteligencia artificial, el cual debe ser capaz de interpretar estos 4 casos distintos de forma precisa.

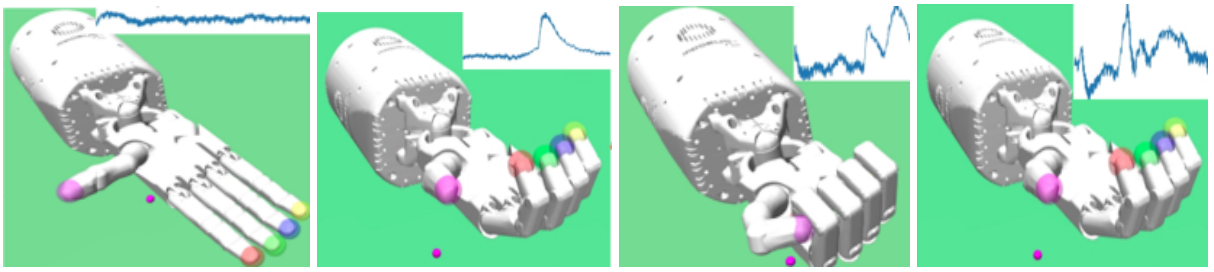

Figura 2.4 La imagen muestra los 4 casos distintos de intención de movimiento con sus respectiva actividad electromiográfica, mano abierta en reposo, transición abierta-cerrada, mano cerrada en reposo, y transición cerrada-abierta.

La figura 2.5 muestra la conexión realizada para la adquisición de datos de electromiografía, esta conexión entre el sensor Myoware, el Arduino Uno y el ordenador se mantiene durante todo el desarrollo de este trabajo, tanto para la parte de adquisición de datos como la parte de control de la mano robótica mediante el modelo de inteligencia artificial.

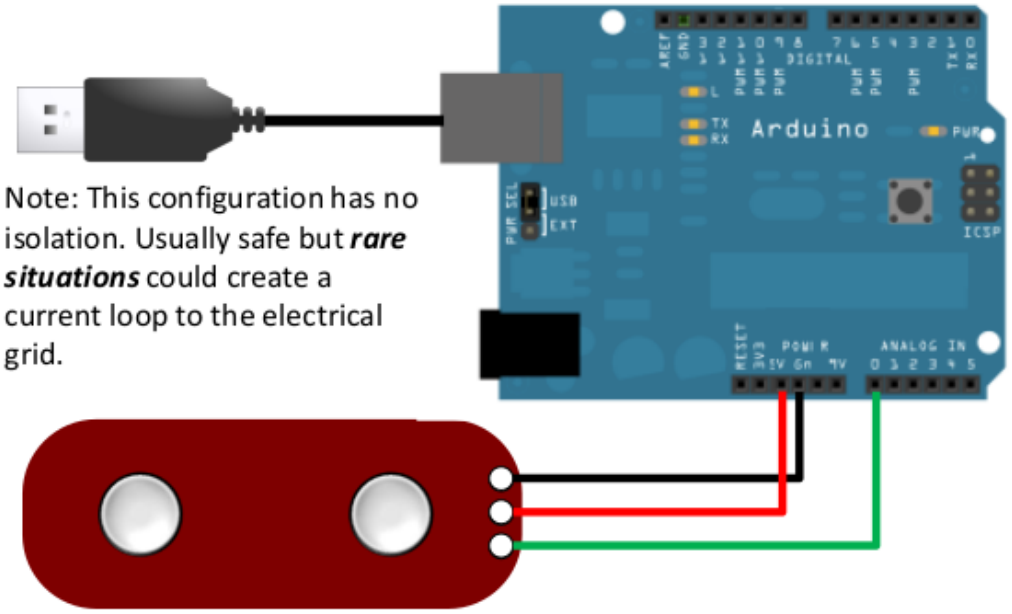

Figura 2.5 Conexión del sistema de control y adquisición de datos

2.2 Entrenamiento del modelo de clasificación

Las redes neuronales se componen de elementos simples operando en paralelo. Estos elementos están inspirados en el sistema nervioso biológico. Así como en la naturaleza, las conexiones entre los elementos determinan en gran parte el funcionamiento de la red. Con el propósito de que la red neuronal realice una función específica esta es entrenada ajustando los valores de las conexiones (pesos) entre los elementos.

Comúnmente, las redes neuronales son ajustadas, o entrenadas, de tal forma que un dato de entrada particular genera como resultado una salida o etiqueta específica. De esta forma se generarán sistemas de reconocimiento de patrones, exponiendo al modelo a ejemplos de pares de entradas y salidas, y ajustando sus parámetros hasta lograr la respuesta deseada por parte del modelo de clasificación.

2.2.1 MATLAB NPRTOOL

Para este trabajo se utilizó el modelo de clasificación de patrones basado en la herramienta *NPRTOOL* de Matlab. Esta herramienta genera una red neuronal sencilla de 3 capas, la capa de entrada, la capa escondida y la capa de salida. La cantidad de parámetros dentro de la capa escondida fue escogida en función del desempeño y la precisión (mayor cantidad de parámetros genera modelos más precisos, sin embargo, consumen más recursos computacionales), y la cantidad de parámetros de la capa de salida está definida por la cantidad de casos posibles que pueden ser seleccionados.

Para entrenar la red neuronal, la herramienta requiere que el conjunto de datos de entrenamiento se presente en un formato específico, un ejemplo del formato se presenta en la figura 2.6. Los datos de entrada pertenecen a las características extraídas de la intención motora representada por las señales de electromiografía adquirida a través de los electrodos del sensor, mientras que los datos de salida pertenecen al valor porcentual que se le da a cada uno de los posibles casos, donde cada columna representa un caso distinto, y cada fila una muestra de la actividad eléctrica al momento de realizar el movimiento perteneciente al caso. Para este trabajo se tomó como característica de los datos de entrada el valor rms de cada uno de los archivos generados al momento de la adquisición de datos, de esta forma cada muestra está representada por un solo valor.

```
in_sample =          out_sample =

  194.4283                1     0     0     0
  175.2813                0     1     0     0
  163.9269                0     0     1     0
  193.4025                0     0     0     1
```

Figura 2.6 Formato de los datos de entrada y de salida requerido para el entrenamiento de la red neuronal. Cada fila representa una muestra y cada columna una clase distinta

Una vez configurados los datos en el formato solicitado por la herramienta, se procedió a realizar el entrenamiento de la red neuronal mediante los siguiente pasos:

1. En la ventana de comandos de Matlab se ejecuto el comando *nnstart*

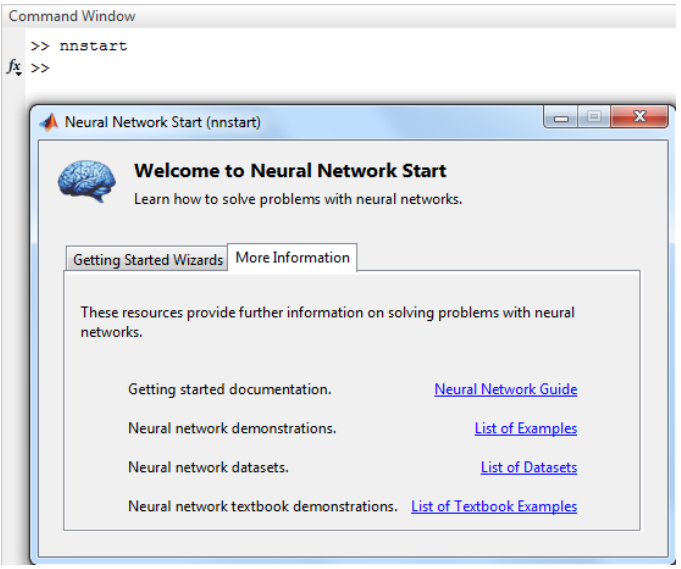

Figura 2.7 GUI de la herramienta de entrenamiento de inteligencia artificial de Matlab

2. En la ventana *Getting Started Wizard* se seleccionó la opción *pattern recognition app*, posteriormente se abrió el GUI de la herramienta *nprtool*

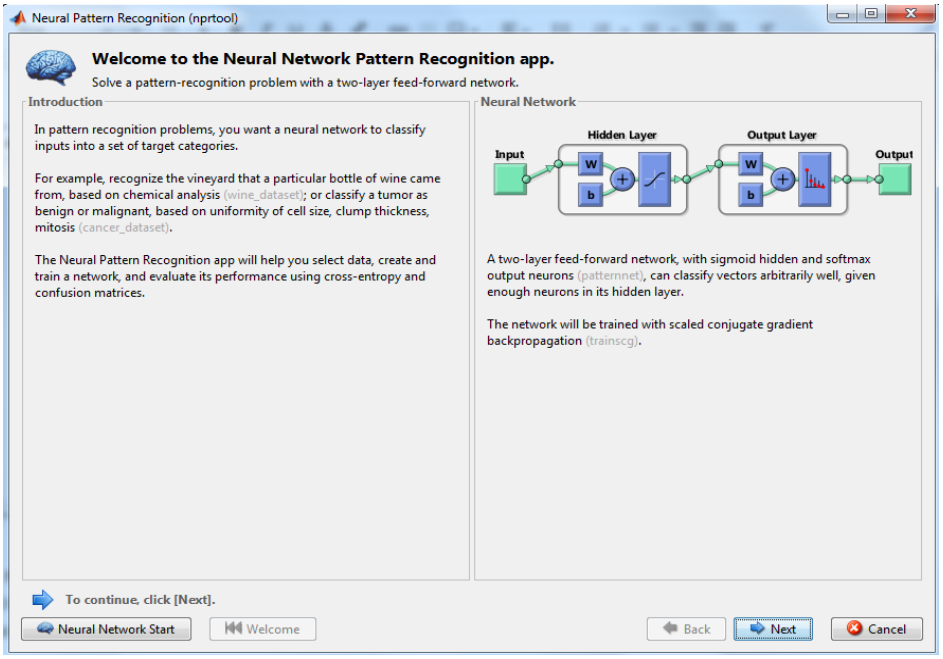

Figura 2.8 Página inicial del GUI de la herramienta NPRTOOL

3. Al dar click en *next*, la siguiente ventana da la opción de especificar los datos de entrada y de salida para entrenar la red neuronal como se muestra en la figura 2.9, para lo cual seleccionamos el conjunto entero de 200 muestras y 4 clases.

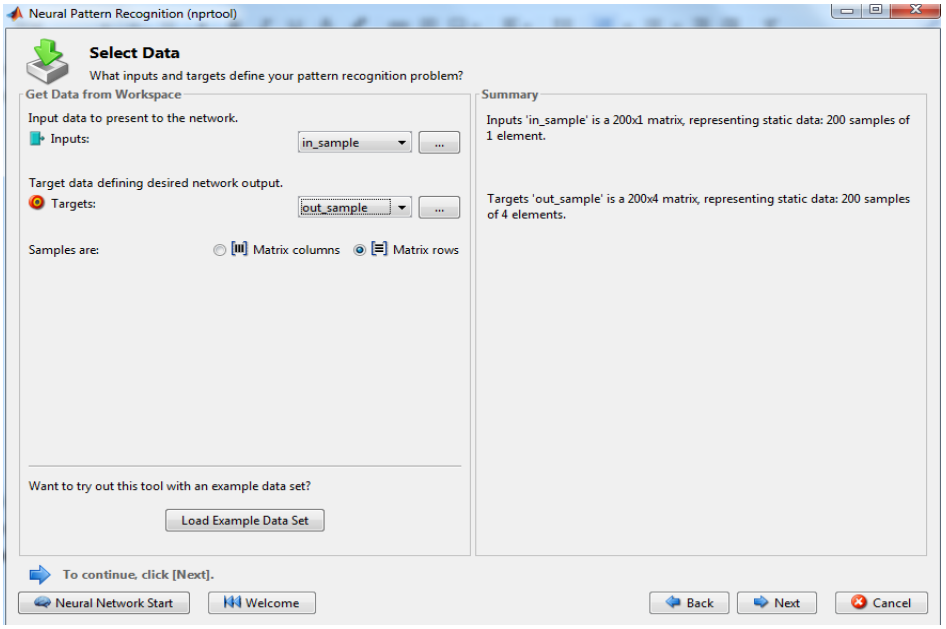

Figura 2.9 Selección de los datos de entrada y de salida para el entrenamiento de la red neuronal

4. Al dar click en *next*, la siguiente página permite la selección de la cantidad de datos que se usarán para entrenamiento, validación, y prueba. Para este trabajo se escogieron los valores predeterminados.

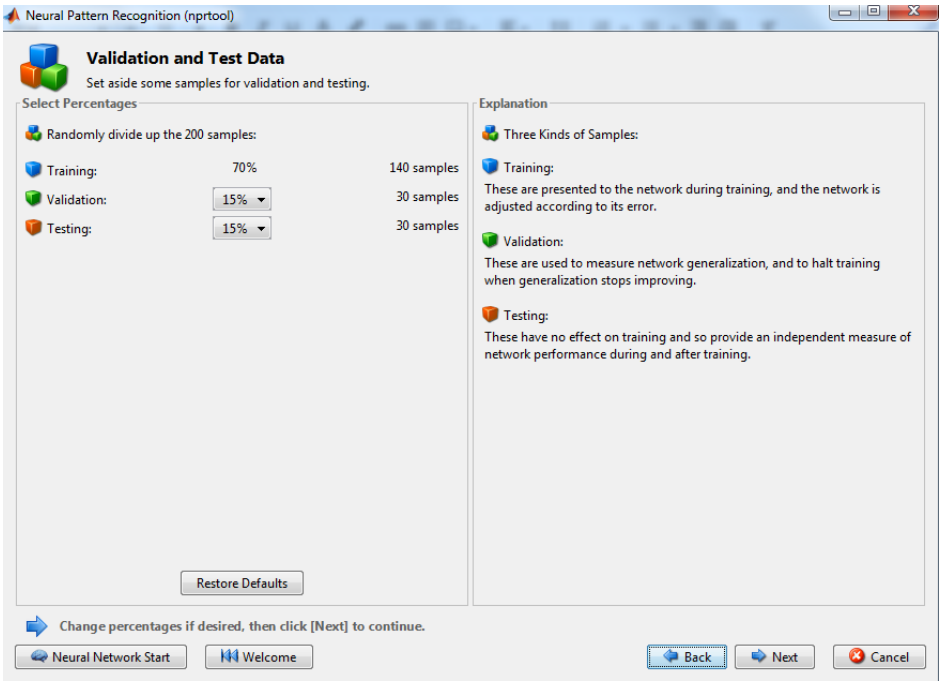

Figura 2.10 Selección de datos de entrenamiento, validación y prueba para el entrenamiento de la red neuronal

5. Al dar click en *next*, en la siguiente ventana se eligio la cantidad de neuronas en la capa escondida, durante el desarrollo de este trabajo se probaron diferentes configuraciones.

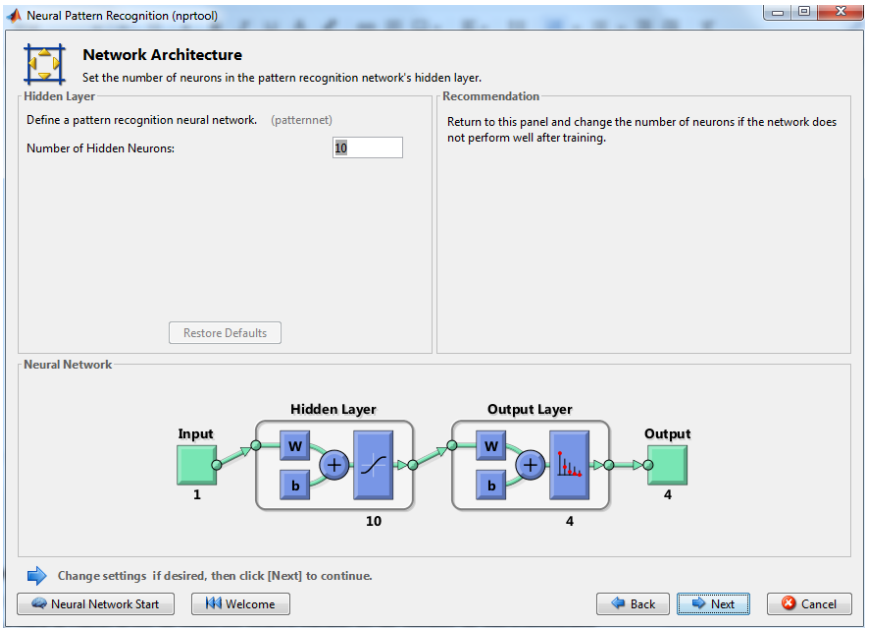

Figura 2.11 Selección de la cantidad de neuronas en la capa escondida de la red neuronal

6. En la ventana siguiente se seleccionó la opción de entrenar, dando como resultado la creación del modelo de clasificación y sus respectivas métricas de desempeño.

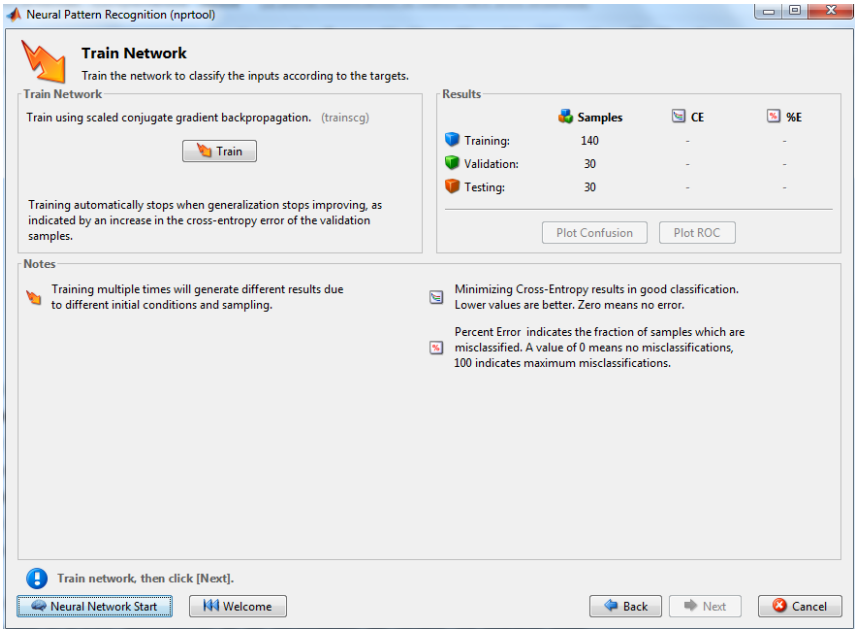

Figura 2.12 Entrenamiento de la red neuronal utilizando las configuraciones previas

7. Al dar click en *next* hasta la ventana final, se seleccionó la opción de generar un diagrama de simulink para la ejecución y visualización del modelo de inteligencia artificial.

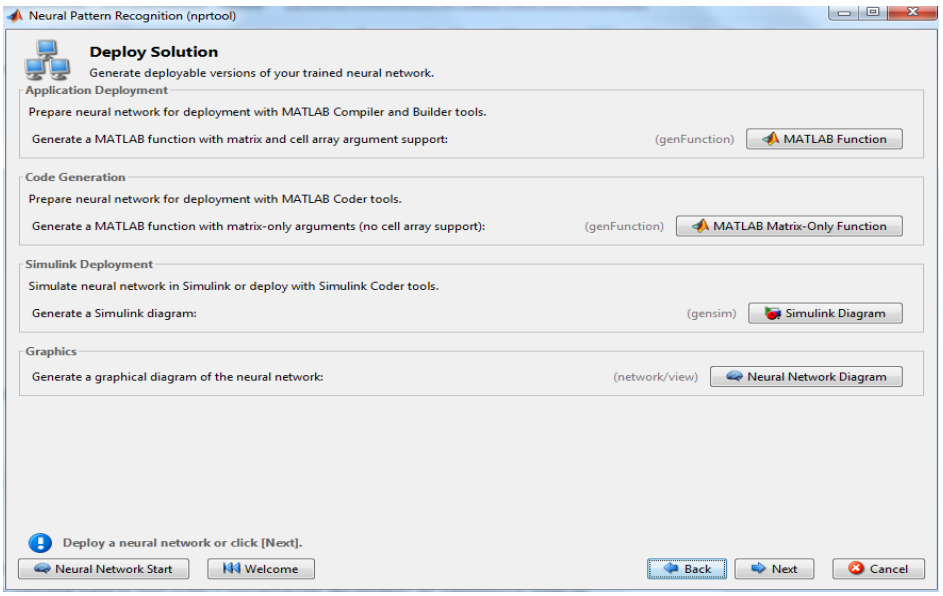

Figura 2.13 Selección del tipo de implementación de la red neuronal entrenada

Una vez realizados los pasos indicados, obtuvimos un modelo de inteligencia artificial, cuyos parámetros fueron optimizados para clasificar cada una de las 200 muestras en 4 clases distintas. El modelo implementado en Simulink corresponde a la figura 2.14, donde se visualizan las 3 capas de la red neuronal. La red neuronal en Simulink nos permite visualizar el funcionamiento del modelo y facilita la implementación manual de la funciones del modelo, al lenguaje de programación de los microcontroladores seleccionados para este trabajo.

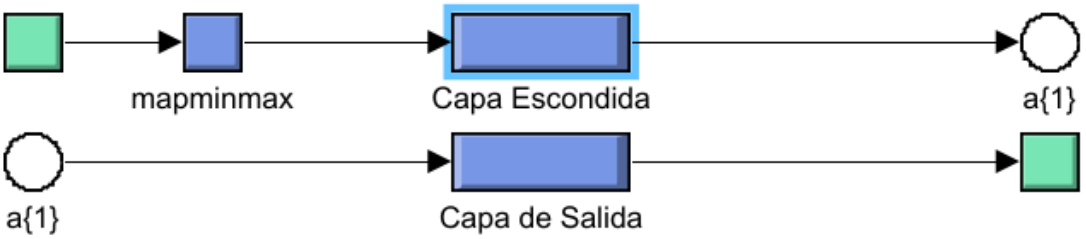

Figura 2.14 Red neuronal entrenada e implementada como modelo de Simulink

2.3 Implementación en sistema embebido

Una vez optimizado el modelo de inteligencia artificial y habiendo obtenido los resultados deseados para la clasificación de intención motora basado en las señales de electromiografía adquiridas al momento de realizar los ejercicios**,** es necesario traducir este modelo desde Simulink hacia C para Arduino, de tal forma que el algoritmo pueda ser ejecutado dentro del sistemas embebido de forma eficiente.

Existen diferentes opciones para traducir modelos de Simulink a otros lenguajes como VHDL,Verilog, C o C++, dependiendo de las necesidades del usuario, sin embargo estas herramientas requieren la compra de licencias adicionales a la licencia básica de Matlab y Simulink, por lo tanto para este trabajo el modelo fue traducido de forma manual mediante el diseño de funciones equivalentes a las implementadas por la herramienta *nprtool*.

Cada bloque implementado en el modelo de Simulink realiza una función básica basada en las estructuras de redes neuronales. El primer bloque *mapminmax*,

procesa los valores de entrada del sistema cambiando el valor máximo y el mínimo por 1 y menos -1 respectivamente, todos los valores intermedios son mapeados de forma proporcional entre estos nuevos límites. Los parámetros de esta función fueron establecidos durante el entrenamiento de la red neuronal, tomando el valor máximo y mínimo del conjunto de datos dando como resultado la ecuación 2.1.

$$Y = \left(\frac{Ymax - Ymin}{Xmax - Xmin}\right)(X - Xmin) - Y \tag{2.1}$$

El siguiente bloque es la capa escondida de la red neuronal, donde se realiza la multiplicación de los pesos por la entrada del sistema y la suma de los sesgos, ambos son parámetros optimizados para la correcta clasificación de los datos durante el entrenamiento de la red neuronal.

$$Tansig = \left(\frac{2}{1 + e^{-2x}}\right) - 1 \tag{2.2}$$

Los sistemas de clasificación basados en inteligencia artificial utilizan funciones no lineales llamadas funciones de activación, las cuales permiten crear relaciones más complejas entre los datos de entrada y de salida del modelo, para esta red neuronal se utilizó la función hiperbólica tangente sigmoidal (ecuación 2.2) como función de activision en la capa escondida

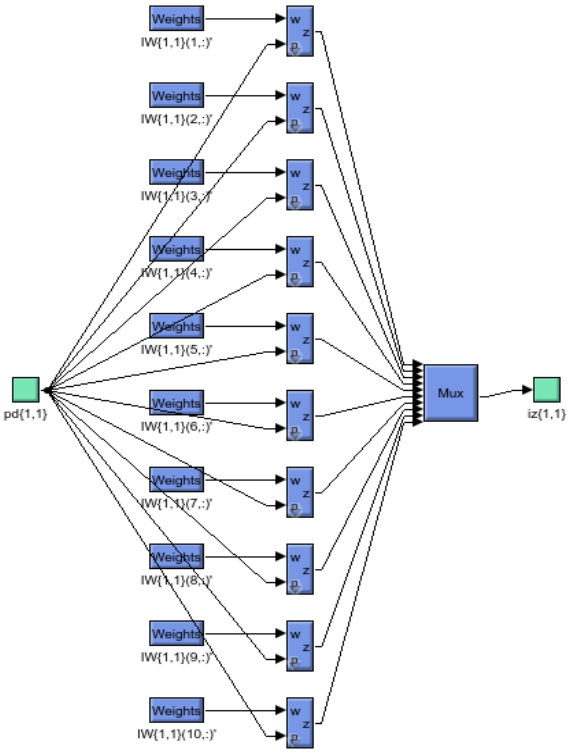

Figura 2.15 Capa escondida de la red neuronal implementada en Simulink

Por último tenemos la capa de salida, el número de neuronas dentro de la capa de salida está determinado por el número de opciones de salida del modelo, para nuestro modelo existen cuatro posibles opciones de las cuales elegir, las acciones estáticas y las acciones de transición. Luego de realizar la multiplicación de los pesos y sesgos de la capa de salida, se aplicó la función de activación *softmax*, con la obtenemos el porcentaje de certeza de que la intención de movimiento sea de reposo o activación.

$$Softmax = \left(\frac{e^{(Xi)}}{\sum_{j=1}^{4} e^{Xi}}\right) \tag{2.3}$$

```c
float *ProcessUnit(float input[],float xmax[],float xmin[],int N)              //Funcion MapMinMax
{                                                                              //N pertence a la cantidad
    int i;                                                                     //De neuronas en la capa escondida
    float ymax=1,ymin=-1;                                                      //Nuevo Rango de salida
    static float output[N];                                                    //Variable de Salida
    for(i=0;i<N;i++)
    {
        output[i]=((ymax-ymin)/(xmax[i]-xmin[i]))*(input[i]-xmin[i])+ymin;
    }
    return output;
}

float tansig(float input1)                                                     //Funcion de activacion
{
    float tansigout=(2/(1+exp(-2*input1))-1);
    return tansigout;
}

float *softmax(float entrada[],int M)                                          //Funcion de activacion
{
    int i=0,j=0;
    static float salida[M];                                                    //Variable de Salida
    float variable1=0;                                                         //Variable Intermedia
    for(i=0;i!=M;++i)
    {
        variable1=(exp(entrada[i])+variable1);
    }
    for(j=0;j!=M;++j)
    {
        salida[j]=exp(entrada[j])/variable1;
    }
    return salida;
}
```

Figura 2.16 Código de las funciones utilizadas en modelo de inteligencia artificial equivalente a la implementación en Simulink en lenguaje C para microcontroladores

Una vez diseñado el programa con las funciones equivalentes a las implementadas en el modelo de inteligencia artificial entrenado en Simulink, el código es evaluado en distintos microcontroladores, para determinar cuál de ellos ofrece la mejor relación entre desempeño,uso de recursos computacionales y precio.

2.4 Ambiente HandReach-V0 como simulación de prótesis activa

El Ambiente HandReach-V0 de la libreria Gym de OpenAI, está basado en la mano robótica Shadow Hand, la cual es una mano robotic antropomórfica con 24 grados de libertad. de estas 24 articulaciones, 20 puede ser controladas independientemente mientras que las restantes son articulaciones acopladas. Las acciones son de 20 dimensiones: se utiliza control de posición absoluta para todas las articulaciones no acopladas de la mano.

```
                                                    #Galo Sanchez Granja

import serial                                       #Com puerto Serial
import gym                                          #GYM
env = gym.make('HandReach-v0')                      #Instancia mano robotica
obs = env.reset()
arduino = serial.Serial('/dev/ttyACM0',9600,timeout=.1)         #Instancia com
pos=-1
while True:
    data = arduino.readline()[:-2]
    print(data)                                     #Para Debug
    if(int(data)==1):                               #Transicion mano abierta a mano cerrada
        for pos in range(-1000,1000,1):             #1000 pasos de -1 a 1
            env.render()
            pos = pos/1000
            obs, reward, ins_done, _ = env.step([0,0,0,pos,pos,0,pos,pos,0,pos,pos,-1,0,pos,pos,0,0,-1,pos,pos])
    elif(int(data)==2):                             #Transicion mano cerrada a mano abierta
        for pos in range(1000,-1000,1):             #1000 pasos de 1 a -1
            env.render()
            pos = pos/1000
            obs, reward, ins_done, _ = env.step([0,0,0,pos,pos,0,pos,pos,0,pos,pos,-1,0,pos,pos,0,0,-1,pos,pos])
    elif(int(data)==0):                             #Mantener la mano es su posicion
        env.render()
        obs, reward, ins_done, _ = env.step([0,0,0,pos,pos,0,pos,pos,0,pos,pos,-1,0,pos,pos,0,0,-1,pos,pos])
env.close()
```

Figura 2.17 Código para la simulación y control de la mano robótica alojada en el ordenador desde el microcontrolador a través del puerto serial

El código mostrado en la figura 2.17 pertenece al utilizado en este trabajo para el manejo del ambiente HandReach para la simulación y control de la prótesis activa. Para el manejo de la prótesis simulada en el ordenador desde los distintos microcontroladores, se utilizó el puerto de comunicación serial por donde se reciben los datos del microcontrolador hacia el programa, los datos entrantes se almacenan en la variable *data*. Durante la ejecución del sistema, el sensor de electromiografía capta la actividad muscular del usuario y el algoritmo de inteligencia artificial alojado en el microcontrolador interpreta la intención motora de forma constante. Si el valor del dato enviado es igual a uno, el programa ejecuta la acción de transición entre mano abierta a mano cerrada, si el valor del dato enviado es igual a dos, el programa ejecuta la acción de transición entre mano cerrada a mano abierta. Por último cuando el algoritmo de inteligencia artificial alojado en el microcontrolador infiere que la intención motora del usuario es mantener la mano abierta, o mantener la mano cerrada, el dato enviado es igual a cero y el programa de control realiza la acción de mantener la prótesis en su posición actual

CAPÍTULO 3

3. Resultados y Análisis

3.1 Red Neuronal

En la tabla 3.1 se muestran los distintos resultados obtenidos en cuanto a la precisión de la red neuronal en función de la cantidad de neuronas en la capa escondida. Al variar la cantidad de neuronas en la capa escondida la diferencia en precisión fue prácticamente despreciable, sin embargo el costo computacional se eleva de forma proporcional ya que aumentan los parámetros del sistema. Estos resultados no fueron satisfactorios ya que un usuario de este sistema está sujeto a fallas en 30 de cada 100 movimientos realizados.

Tabla 3.1 Precisión del modelo de clasificacion en funcion de la cantidad de neuronas en la capa escondida

Neuronas en la capa escondida	Precisión del clasificador
5	69.5%
10	71.5%
20	64.5%
50	71.5%
100	65%

En la figura 3.1, se muestra una gráfica de confusión, la cual se lee de la siguiente manera: las filas corresponden a la clase predecida por el modelo de clasificación, y las columnas corresponden a la clase real especificada dentro de los datos de entrenamiento (etiquetas). Las celdas diagonales corresponden a las observaciones que han sido clasificadas de forma correcta. Todas las celdas fuera de la diagonal corresponden a las observaciones que han sido clasificadas de forma incorrecta. Dentro de cada celda se muestra la cantidad de observaciones y el porcentaje del número total de observaciones.

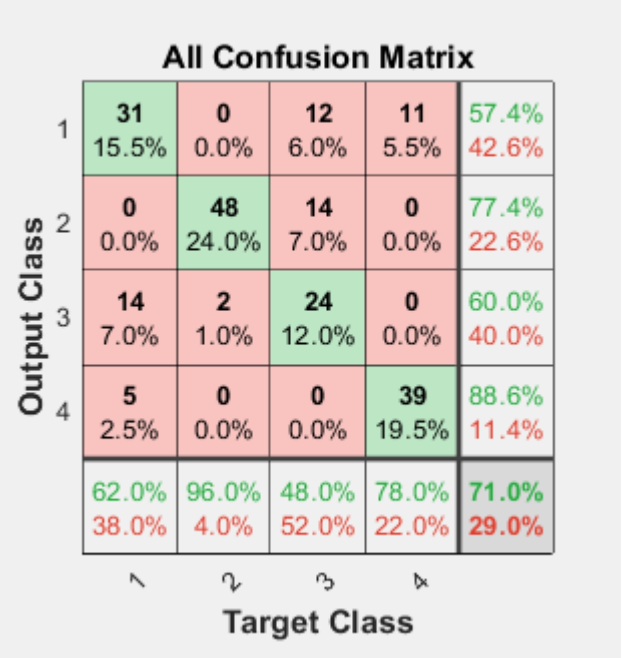

Figura 3.1 Gráfica de confusión de la red neuronal implementada con 10 neuronas en la capa escondida

El modelo de 4 clases mostró problemas al momento de interpretar la intención motora del usuario, en especial entre las clases de transición. Como solución a este problema de bajo desempeño se propuso en este trabajo la implementación de dos modelos distintos de intención de movimientos cada uno de 2 clases, es decir una red neuronal para interpretar si el usuario desea mantener la mano abierta o desea realizar la transición entre mano abierta a mano cerrada, y otra para interpretar si el usuario desea mantener la mano cerrada o si desea realizar la transición entre mano cerrada a mano abierta. Debido a que las transiciones sólo pueden ocurrir dentro de clases estáticas, cada una de las redes neuronales se desactiva cuando la otra está funcionando. Para esta nueva implementación se repitieron los pasos expuestos en el capítulo 2 para el entrenamiento de redes neuronales en matlab, y la transcripción del código al lenguaje del microcontrolador, los resultados se muestran en la figura 3.2.

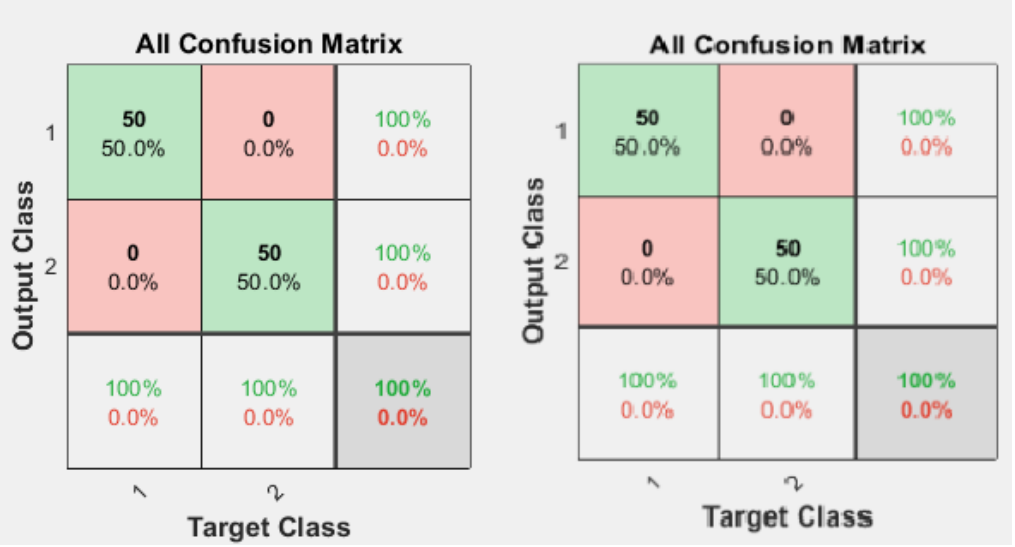

Figura 3.2 Muestra dos gráficas de confusión para dos modelos de clasificación de intención motora basado en la adquisición de señales mioeléctricas

Este nuevo acercamiento género un sistema capaz de clasificar con una precisión del 100% todas las clases distintas dentro de los datos tomados para este trabajo, utilizando tan solo 5 neuronas en la capa escondida, de esta forma se alcanzó el desempeño deseado en el sistema utilizando pocos recursos computacionales.

3.2 Microcontroladores

La tabla 3.2 muestra los resultados obtenidos al momento de evaluar el algoritmo de inteligencia artificial en los 3 microcontroladores propuestos para este trabajo. Los resultados son consistentes con las características especificadas por los fabricantes. El Arduino UNO mostró un mejor desempeño en cuanto a tiempo de ejecución del algoritmo, siendo 2 veces más rápido que el Arduino Nano y 480 veces más rápido que el MCU ESP8266. La cantidad de memoria consumida en cada microcontrolador varía a pesar de tratarse del mismo programa, esto es debido a la diferencias de arquitectura que existen en los procesadores, esto da como resultado que el MCU ESP8266 utiliza más memoria para ejecutar el mismo programa que los demás microcontroladores.

Tabla 3.2 Desempeño y uso de recursos computacionales de cada uno de los microcontroladores utilizados en este trabajo

Microcontrolador	Tiempo de ejecución	Almacenamiento de programa	Memoria dinámica
Arduino UNO	4 us	5 %	9 %
Arduino NANO	8 us	6 %	2 %
ESP8266	1920 us	25 %	32 %

Una vez obtenidos los resultados del desempeño de cada uno de los microcontroladores bajo el uso del mismo algoritmo de inteligencia artificial, queda claro que aquel que ofrece un mejor desempeño es el Arduino UNO, sin embargo en cuenta a la relación entre el costo y el desempeño de los microcontroladores, no existe un claro ganador, dado que el Arduino Uno ofrece dos veces el desempeño del Arduino Nano pero de igual forma al doble del precio ($20 y $10 respectivamente). Basado en el criterio de portabilidad se sugiere el uso del Arduino Nano para este trabajo, debido a que para sistemas que el usuario debe vestir se prefieren dispositivos de menor escala.

3.3 Costos

Durante el desarrollo de este trabajo se utilizaron varios programas con licencia y se compraron distintos dispositivos, el detalle de cada uno de los dispositivos y programas usados con sus respectivos precios se encuentran en la tabla 3.3. Además se tomó en cuenta el costo por el desarrollo, prueba y validación de los programas de inteligencia artificial, adquisición de datos y control de prótesis, basado en el salario promedio que recibe un ingeniero realizando trabajos similares a los largo de un periodo de tiempo de 4 meses. Cabe recalcar que para el desarrollo de este trabajo se solicitó una licencia de prueba para la librería Gym de OpenAI el cual utiliza el programa Mujoco para los ambientes de simulación de la categoría robótica, la licencia anual académica de este programa tiene un valor de $3000.

Tabla 3.3 Costos para el desarrollo del trabajo presentado

Detalle	Costo
Arduino Uno	$20
Arduino Nano	$10
ESP8266	$6
Myoware	$40
Electrodos	$30
cables varios	$5
Licencia estudiantil anual Matlab/Simulink	$80
Licencia de prueba GYM/Mujoco	$0
Trabajo de ingeniería	$3200

CAPÍTULO 4

4. Conclusiones y recomendaciones

En el desarrollo de esta trabajo se utilizaron varios recursos con el fin de diseñar un sistema de control de prótesis activa basado en la precisa interpretación de la intención motora utilizando señales mioeléctricas, funcional con el fin de mejorar la calidad de vida de las personas amputadas, y de bajo costo para poder crear un producto que pueda llegar a varios sectores de la población que no tiene acceso a este tipo de dispositivos que se ofertan actualmente en el mercado

4.1 Conclusiones

- Utilizando dispositivos de bajo costo y programas de código abierto, se pudo diseñar e implementar un sistema que es capaz de interpretar la intención motora del usuario por medio de la adquisición de señales mioeléctricas a través de electrodos situados en la zona del antebrazo, esta información mostró ser suficiente al igual que la cantidad de recursos computacionales disponibles en las tres plataformas propuestas para alojar el sistema de clasificación basado en redes neuronales.
- Para entrenar el sistema de clasificación basado en redes neuronales, se creó un conjunto de datos de 200 muestras, compuesto de 50 muestras por cada uno de los 4 casos distintos de intención de movimiento, como resultado se obtuvo un modelo de clasificación cuya precisión es del 100% al momento de clasificar los datos tomados para el entrenamiento.
- El modelo de clasificación basado en redes neuronales e implementado en Simulink fue la base para el diseño del algoritmo de clasificación ejecutado en los microcontroladores de este trabajo, mediante el diseño de funciones y operaciones equivalentes a las utilizadas en Simulink, el programa es capaz de generar los mismo resultados que el modelo genera en el ordenador haciendo uso eficiente de los recursos computacionales disponibles en los sistemas embebidos.
- Una vez obtenidos los resultados del desempeño de cada uno de los microcontroladores bajo el uso del mismo algoritmo de inteligencia artificial, queda claro que aquel que ofrece un mejor desempeño es el Arduino UNO, sin

embargo en cuenta a la relación entre el costo y el desempeño de los microcontroladores, no existe un claro ganador, dado que el Arduino Uno ofrece dos veces el desempeño del Arduino Nano pero de igual forma al doble del precio ($20 y $10 respectivamente).

3.2 Recomendaciones

- Se recomienda incrementar la cantidad de muestras utilizadas para generar el conjunto de datos de entrenamiento del modelo de clasificación, a pesar de haber obtenido buenos resultados con tan solo 200 muestras existe la oportunidad de tomar muestras durante distintas actividades físicas que puedan cambiar el comportamiento de los datos adquiridos.
- El uso de librerías de código abierto como pytorch, para el diseño de sistemas de inteligencia artificial, pueden reducir significativamente el costo generado por el uso de programas con licencia, además de que son fáciles de usar y pueden generar sistemas aún más flexibles y robustos.
- Se recomienda explorar otros casos de intención de movimiento, para así utilizando el mismo método que el propuesto en este trabajo, generar sistemas de interpretación de intención motora los cuales puedan ampliar la cantidad de activa que una persona amputada puede realizar utilizando una prótesis activa

BIBLIOGRAFÍA

[1] Castellini, C. and van der Smagt, P., 2008. Surface EMG in advanced hand prosthetics. *Biological Cybernetics*, 100(1).

[2] D. Bright, A. Nair, D. Salvekar and S. Bhisikar, "EEG-based brain controlled prosthetic arm," *2016 Conference on Advances in Signal Processing (CASP)*, Pune, India, 2016, pp. 479-483, doi: 10.1109/CASP.2016.7746219.

[3] S. Sudarsan, E. Chandra Sekaran,Design and Development of EMG Controlled Prosthetics Limb,Procedia Engineering,Volume 38,2012,Pages 3547-3551,ISSN 1877-7058

[4] Akshar Joshi and Gautam Mishra. 2010. Artificial intelligence. In Proceedings of the International Conference and Workshop on Emerging Trends in Technology. Association for Computing Machinery, New York, NY, USA, 1023. DOI:https://doi.org/10.1145/1741906.1742236

[5] Géron, A., 2020. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. Beijing: O'Reilly.

[6] Dan Gunnarsson, Stefan Kuntz, Glenn Farrall, Akihito Iwai, and Rolf Ernst. 2012. Trends in automotive embedded systems. In Proceedings of the 2012 international conference on Compilers, architectures and synthesis for embedded systems. Association for Computing Machinery, New York, NY, USA, 9–10. DOI:https://doi.org/10.1145/2380403.2380410

[7] George Neville-Neil. 2003. The Truth About Embedded Systems: Embedded systems are different in several ways from other software environments. 1, 2 (April 2003), 4–5. DOI:https://doi.org/10.1145/644254.644260

[8] Lobo-Prat, J., Keemink, A.Q., Stienen, A.H. et al. Evaluation of EMG, force and joystick as control interfaces for active arm supports. J NeuroEngineering Rehabil 11, 68 (2014). https://doi.org/10.1186/1743-0003-11-68

[9] A. Pradhan et al., "Acquisition and classification of EMG using a dual-channel EMG biopotential amplifier for controlling assistive devices," 2016 IEEE Annual India Conference (INDICON), Bangalore, India, 2016, pp. 1-5, doi: 10.1109/INDICON.2016.7839015.

[10] K. Kiguchi and Y. Hayashi, "An EMG-Based Control for an Upper-Limb Power-Assist Exoskeleton Robot," in IEEE Transactions on Systems, Man, and

Cybernetics, Part B (Cybernetics), vol. 42, no. 4, pp. 1064-1071, Aug. 2012, doi: 10.1109/TSMCB.2012.2185843.

[11] Cheesborough, J. E., Smith, L. H., Kuiken, T. A., & Dumanian, G. A. (2015). Targeted muscle reinnervation and advanced prosthetic arms. Seminars in plastic surgery, 29(1), 62–72. https://doi.org/10.1055/s-0035-1544166

[12] Lapan, M., 2020. *Deep reinforcement learning hands-on*. Packt Publishing. ISBN: 9781838826994

[13] Bettina Weiss, Günther Gridling, and Markus Proske. 2005. A case study in efficient microcontroller education.2, 4 (October 2005), 40–47. DOI:https://doi.org/10.1145/1121812.1121821

[14] Store.arduino.cc. 2021. *Arduino Uno Rev3 | Arduino Official Store*. [online] Available at: <https://store.arduino.cc/usa/arduino-uno-rev3> [Accessed 23 February 2021].

[15] Store.arduino.cc. 2021. *Arduino Nano Every | Arduino Official Store*. [online] Available at: <https://store.arduino.cc/usa/nano-every> [Accessed 23 February 2021].

[16] Espressif.com. 2021. *ESP8266 Wi-Fi MCU I Espressif Systems*. [online] Available at: <https://www.espressif.com/en/products/socs/esp8266> [Accessed 23 February 2021].

[17] Gym.openai.com. 2021. *Gym: A toolkit for developing and comparing reinforcement learning algorithms*. [online] Available at: <https://gym.openai.com/> [Accessed 23 February 2021].

[18] Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., Kumar, V., & Zaremba, W. (2018). Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research. ArXiv, abs/1802.09464.

[19] Andrychowicz, O.M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J.W., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., & Zaremba, W. (2020). Learning dexterous in-hand manipulation. The International Journal of Robotics Research, 39, 20 - 3.

[20] Andrychowicz, M., Crow, D., Ray, A., Schneider, J., Fong, R.H., Welinder, P., McGrew, B., Tobin, J., Abbeel, P., & Zaremba, W. (2017). Hindsight Experience Replay. NIPS.

[21]GitHub. 2021. AdvancerTechnologies/MuscleSensorV3. [online] Available at:<https://github.com/AdvancerTechnologies/MuscleSensorV3/blob/master/Documents/Muscle%20Sensor%20v3%20Schematic.pdf> [Accessed 23 February 2021].
[22]GitHub. 2021. AdvancerTechnologies/MyoWare_MuscleSensor. [online] Available at:<https://github.com/AdvancerTechnologies/MyoWare_MuscleSensor/blob/master/Documents/AT-04-001.pdf> [Accessed 23 February 2021].