



UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE TECNOLOGIA E INGENIERIA ELECTRONICA

DISPOSITIVO DE CONTROL A PARTIR DE SEÑALES ELECTROMIOGRAFICAS

Por:

Kenny Alejandro Jaimes Manrique

PROYECTO DE GRADO

Presentado ante la Ilustre Universidad Simón Bolívar
como requisito parcial para optar al título de
Ingeniero Electrónico

Sartenejas, Septiembre de 2018



UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE TECNOLOGIA E INGENIERIA ELECTRONICA

DISPOSITIVO DE CONTROL A PARTIR DE SEÑALES ELECTROMIOGRAFICAS

Por:

Kenny Alejandro Jaimes Manrique

Realizado con la asesoría de:

Gerardo Fernández-López

PROYECTO DE GRADO

Presentado ante la Ilustre Universidad Simón Bolívar
como requisito parcial para optar al título de
Ingeniero Electrónico

Sartenejas, Septiembre de 2018



UNIVERSIDAD SIMÓN BOLÍVAR
VICERRECTORADO ACADÉMICO
DECANATO DE ESTUDIOS PROFESIONALES
Coordinación de Ingeniería y Tecnología Electrónica

ACTA DE EVALUACION DE PROYECTO DE GRADO

Código de la asignatura: EP5406

Fecha: 10-10-2018

Nombre del estudiante: Kenny Alejandro Jaimes

Carnet: 11-10486

Manrique

Título del proyecto: Dispositivo de control a partir de señales electromiográficas

Tutor: Prof. Gerardo Fernández López

Jurados: Profesores Gerardo Ceglia Diotaiuti y Guillermo Villegas

☒ APROBADO

☐ REPROBADO


Observaciones:


El jurado examinador, **por unanimidad**, considera el proyecto de grado merecedor de la mención especial SOBRESALIENTE:

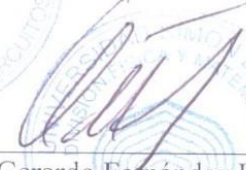
☐ SI

☒ NO

En caso afirmativo, justifique su decisión en estricto cumplimiento del documento "Criterios para otorgar la mención especial en proyectos de grado y pasantías largas e intermedias" (ver al dorso):


Prof. Gerardo Ceglia Diotaiuti
Jurado
C.I. 6.874.439


Prof. Guillermo Villegas
Jurado
C.I. 3.637.152


Prof. Gerardo Fernández López
Tutor Académico
C.I. 5.134.431

Prof. Nombres y Apellidos
Co-Tutor
C.I.

Prof. Nombres y Apellidos
Jurado
C.I.

Nota: Colocar los sellos de los respectivos Departamentos Académicos. Para jurados externos usar el sello de la Coordinación Docente. Este documento debe entregarse sin enmiendas.



UNIVERSIDAD SIMÓN BOLÍVAR
VICERRECTORADO ACADÉMICO
DECANATO DE ESTUDIOS PROFESIONALES
Coordinación de Ingeniería y Tecnología Electrónica

ACTA DE EVALUACION DEL INFORME DE PROYECTO DE GRADO

Código de la asignatura: EP5406

Fecha: 10-10-2018

Nombre del estudiante: Kenny Alejandro Jaimes
Manrique

Carnet: 11-10486

Título del proyecto: Dispositivo de control a partir de señales electromiográficas

Tutor: Prof. Gerardo Fernández López


Jurados: Profesores Guillermo Villegas y Gerardo Ceglia


Nosotros, miembros del Jurado designado por la Coordinación Docente de Tecnología e Ingeniería Electrónica, en la fecha indicada para la evaluación del Proyecto en cuestión, hemos evaluado el contenido del mismo y hacemos constar nuestra decisión de **aceptar el documento para su presentación en acto público**. Se convoca al estudiante a la presentación pública del Proyecto en el edificio "Electrónica" sala "323".


Fecha: 10/10/2018

Hora: 01:30 pm

Lugar: Sede de Sartenejas


Prof. Gerardo Ceglia Diotaiuti
Jurado (Presidente)
C.I. 6.874.439


Prof. Guillermo Villegas
Jurado
C.I. 3.637.152


Prof. Gerardo Fernández López
Tutor
C.I. 5.134.431

Nota: Colocar los sellos de los respectivos Departamentos Académicos. Para jurados externos usar el sello de la Coordinación Docente. Este documento debe entregarse en original y sin enmiendas.



UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE TECNOLOGIA E INGENIERIA ELECTRONICA

DISPOSITIVO DE CONTROL A PARTIR DE SEÑALES ELECTROMIOGRAFICAS
PROYECTO DE GRADO

Realizado por: Kenny Alejandro Jaimes Manrique

Con la asesoría de: Gerardo Fernández-López

RESUMEN

En el presente documento se explica el diseño e implementación del prototipo de un dispositivo de control a partir del reconocimiento de gestos realizados con las manos. Con este dispositivo, se busca crear un control especialmente flexible que permita su utilización en diferentes situaciones. Dicho dispositivo consiste en un conjunto de sistemas de acondicionamiento, adquisición, procesamiento y análisis de señales electromiográficas provenientes de los músculos ubicados en el antebrazo. Esto permite la interpretación de gestos con ayuda de un algoritmo de aprendizaje de máquinas. El circuito de acondicionamiento se diseñó luego de un estudio detallado de las características y propiedades de dichas señales, que permite que el proceso adquisición sea lo suficientemente robusto para responder en variadas situaciones. Haciendo uso del microprocesador MK22F de NXP, se realiza la adquisición y el análisis de las señales, esto último comprende la extracción de características a partir de algoritmos de descripción aplicables a este tipo de señales. Dichas características alimentan una red neuronal artificial implementada dentro del mismo microprocesador. Dicha red fue diseñada en MATLAB y es capaz de reconocer un conjunto específico de gestos a partir de un entrenamiento supervisado. La finalidad de esta detección de gestos es permitir el control de aplicaciones en diferentes terminales (teléfonos inteligentes, computadores, consolas de videojuegos, tabletas, etc...). Como prueba del funcionamiento del sistema en tiempo real, se desarrolló un programa de demostración en el motor de videojuegos Unity3D.

Palabras clave: señal electromiográfica, procesamiento de señales, red neuronal artificial.

INDICE GENERAL

RESUMEN	v
INTRODUCCION.....	1
CAPITULO 1	4
MARCO TEORICO.....	4
1.1 Señales electromiográficas	4
1.2 Electroodos.....	5
1.3 Aprendizaje de maquinas.....	5
1.4 Red neuronal artificial espacio arriba	6
1.4.1 Redes neuronales prealimentadas.....	6
1.4.2 Redes neuronales recurrente.....	7
1.4.3 Tipos de neuronas espacio arriba	8
1.4.1 Función de error: Entropía cruzada.....	9
1.4.2 Algoritmo de aprendizaje: Propagación hacia atrás	11
CAPITULO 2	12
DISEÑO CONCEPTUAL	12
2.1 Preprocesamiento de la señal	12
2.2 Procesamiento de las señales	13
2.3 Detección de gestos	15
2.4 Programa de demostración.....	16
CAPITULO 3	17
SISTEMA DE ADQUISICION Y ACONDICIONAMIENTO	17
3.1 Electroodos.....	17
3.1.1 Diseño.....	17
3.1.2 Implementación	20
3.2 Amplificación.....	23
3.2.1 Diseño.....	23
3.2.2 Circuito de preamplificación	26
3.2.3 Segunda etapa de amplificación	29
3.3 Filtros.....	30
3.3.1 Diseño.....	30
3.3.2 Filtro pasa alto Switched capacitor	32

3.3.3 Filtro pasa bajo	34
3.4 Rectificación	36
3.5 Alimentación	38
3.6 Sistema de acondicionamiento entero	40
3.7 Resultados de la adquisición	44
CAPITULO 4	48
PROCESAMIENTO DE LA SEÑAL.....	48
4.1 Adquisición y guardado de datos	48
4.2 Extracción de características	51
4.2.1 Media de la señal	53
4.2.2 Varianza.....	53
4.2.3 Cambios de signo de la pendiente	53
4.2.4 Longitud de forma de onda (Waveform Lenght)	54
4.2.5 Amplitud de Willison.....	55
CAPITULO 5	57
RED NEURONAL.....	57
5.1 Entrenamiento	57
5.2 Implementación	64
5.3 Resultados del entrenamiento de la red neuronal	69
CAPITULO 6	72
PROGRAMA DE DEMOSTRACION	72
CONCLUSIONES	77
RECOMENDACIONES	79
REFERENCIAS.....	81
APENDICES	84
Apéndice A: Código para extracción de características.	84
Apéndice B: Código de funciones de la red neuronal.	88
Apéndice C: Código principal del dispositivo.	91
Apéndice D: Código para entrenamiento de la red neuronal	99
Apéndice E: Código del programa de ejemplo.....	101

INDICE DE FIGURAS

Figura 1.1 Composición de señal EMG [4].	4
Figura 1.2 Red neuronal prealimentada.	7
Figura 1.3 Red neuronal recurrente.	8
Figura 1.4 Diagrama de red neuronal con bias.	9
Figura 1.5 Función logarítmica.	10
Figura 2.1 Diagrama general del sistema.	12
Figura 2.2 Diagrama del preprocesamiento de la señal	13
Figura 2.3 Diagrama de adquisicion de datos	14
Figura 2.4 Diagrama del procesamiento de la señal	14
Figura 2.5 Diagrama de la detección de gestos	16
Figura 3.1 Ejemplo de electrodo de Ag/AgCl [16].	18
Figura 3.2 Ejemplo de electrodo de acero inoxidable [16].	19
Figura 3.3 Ejemplo de electrodo orbital [16].	19
Figura 3.4 Electrodo de Ag/AgCl	20
Figura 3.5 Cables de los electrodos	21
Figura 3.6 Músculos medidos.	21
Figura 3.7 Medición del músculo braquiorradial.	22
Figura 3.8 Medición del músculo flexor cubital del carpo.	22
Figura 3.9 Electrodo de referencia en el codo.	23
Figura 3.10 diagrama de pre-amplificación de la señal	25
Figura 3.11 Amplificador de instrumentación INA121 [19].	26
Figura 3.12 Amplificador de instrumentación con retroalimentación implementado.	28
Figura 3.13 Amplificador no inversor implementado.	29
Figura 3.14 Comparación de filtros	31
Figura 3.15 Filtro pasa alto Sallen Key implementado.	32
Figura 3.16 Señal de control (FTM).	35
Figura 3.17 Filtro pasa bajo MAX7480.	36
Figura 3.18 Circuito de rectificación de onda completa [24].	37
Figura 3.19 Diagrama del rectificador para polarización positiva [24].	37

Figura 3.20 Diagrama del rectificador para polarización negativa [24].	38
Figura 3.21 Regulador positivo L7805 [25].	39
Figura 3.22 Regulador negativo L7905 [26].	39
Figura 3.23 Circuito de alimentación.	39
Figura 3.24 Diagrama del sistema de acondicionamiento.	41
Figura 3.25 Diagrama del circuito de acondicionamiento – 1 ^{ra} parte.	42
Figura 3.26 Diagrama del circuito de acondicionamiento – 2 ^{da} parte.	43
Figura 3.27 Salida del Amplificador de instrumentación – 10mV/div- 10ms/div.	44
Figura 3.28 Salida del filtro pasa bajo – 10mV/div- 10ms/div.	45
Figura 3.29 Salida del amplificador – 1V/div- 10ms/div.	45
Figura 3.30 Salida del rectificador – 1V/div- 10ms/div.	46
Figura 3.31 Salida del filtro pasa bajo – 1V/div- 10ms/div.	46
Figura 4.1 Ventana de tiempo mínima.	49
Figura 4.2 Diagrama de adquisición de datos.	51
Figura 4.3 Ejemplo de arreglos auxiliares.	54
Figura 5.1 Matrices de entrenamiento.	58
Figura 5.2 Ejemplo de matriz de resultados para entrenamiento.	59
Figura 5.3 5 características siendo recibidas por serial.	60
Figura 5.4 Ejemplo de características guardadas en archivo “.CSV”.	61
Figura 5.5 Ejemplo de matriz de entrada para entrenamiento.	61
Figura 5.6 Ejemplo de datos de salida para entrenamiento.	62
Figura 5.7 Ejemplo de matriz de salidas para entrenamiento	62
Figura 5.8 Diagrama de la red neuronal diseñada.	63
Figura 5.9 Diagrama del proceso de entrenamiento.	64
Figura 5.10 Vector de entrada.	65
Figura 5.11 Matriz con pesos entrada - capa oculta.	65
Figura 5.12 Matriz con los pesos capa oculta - salida.	66
Figura 5.13 Función Tansig [17].	66
Figura 5.14 Diagrama del cálculo de las 10 neuronas de la capa oculta.	67
Figura 5.15 Función Softmax [18].	68
Figura 5.16 Diagrama del cálculo de las 4 salidas.	68
Figura 5.17 Resultado con mano descansando.	69

Figura 5.18 Resultado con mano hacia la derecha.	70
Figura 5.19 Resultado con mano hacia la izquierda.	70
Figura 5.20 Resultado con mano abierta.	71
Figura 6.1 Modelos de la mano.	73
Figura 6.2 Mano en descanso.	73
Figura 6.3 Mano abierta.	73
Figura 6.4 Mano hacia la izquierda.	73
Figura 6.5 Mano hacia la derecha.	73
Figura 6.6 Mano hacia la descansando en programa en ejecución.	74
Figura 6.7 Mano hacia la izquierda en programa en ejecución.	75
Figura 6.8 Mano hacia la derecha en programa en ejecución.	75
Figura 6.9 Mano abierta en programa en ejecución.	76

INTRODUCCION

El ser humano durante toda su historia ha demostrado su ingenio en la forma como ha sido capaz de lograr objetivos que en algún momento eran o se pensaban imposibles. Mucho del desarrollo intelectual viene de la mano con el desarrollo de herramientas que nos permiten lograr objetivos de forma más sencilla y que esto a su vez permite solucionar problemas.

Las personas en el mundo actual nos vemos rodeados por aparatos electrónicos que van evolucionando y haciéndose más complejos en su interior pero que a visión del usuario cumplen con hacer la vida más sencilla. Dispositivos tan variados que su utilidad que van desde el entretenimiento como televisores, equipos de sonido, tabletas, consolas de videojuegos; pasando por dispositivos que nos permiten ser más productivos como computadoras personales, teléfonos inteligentes; o incluso sistemas de necesidades básicas como neveras o la iluminación de un hogar que evolucionan con la tecnología pasando a ser más complejos sí, pero también más útiles.

Con todo esto nos encontramos en hogares, oficinas u otros ambientes rodeados de dispositivos que controlamos de diferentes maneras y que de hecho, necesitamos de dispositivos específicamente diseñados para controlar al resto.

En el presente proyecto se plantea el desarrollo de un prototipo que permite controlar diferentes dispositivos, a partir de un concepto simple de entender, control a partir de gestos realizados con la mano. Aunque como se mencionó, existen dispositivos que tienen como principal objetivo controlar al resto de aparatos y hacerlo de forma intuitiva a los usuarios, se buscó plantear una alternativa que sea distinta a lo convencional y que permita realizar actividades que no se pueden con el resto.

Se tienen controles universales con los cuales podemos controlar televisores, equipos de sonido, entre otros, presionando botones; sistemas que trabajan con cámaras y que a partir del procesamiento de imágenes describen nuestro movimiento corporal (o parte de este) para, por ejemplo, interactuar con objetos virtuales en un videojuego como Kinect [1] o Leap Motion [2]; y muchas otras alternativas de control tanto similares, como por ejemplo Myoband [3], así como alternativas totalmente distintas al dispositivo planteado. Aunque la existencia de todas las alternativas se tomaron en cuenta al momento de plantear un dispositivo de control que fuese no

solo sencillo sino que presentara una alternativa que tuviese sentido, lo cierto es que todas presentan limitaciones.

Es por esto que el corazón de este proyecto no es solo lograr controlar algo con gestos (lo cual se sabe que es posible) sino lograr un sistema de reconocimiento de gestos que sea especialmente robusto, capaz de adaptarse a cada usuario y que no esté limitado por las situaciones. Los sistemas más similares al propuesto, tienen una similitud y es que a día de hoy su utilidad es específica para un producto, una situación o incluso los más flexibles presentan al usuario solo con un grupo de controles que hacen que su utilidad final que aunque en teoría es amplia, no lo sea en realidad. A nivel conceptual el dispositivo debía ser fácil de usar y entender, presentar un precio económico frente a la competencia inmediata y más importante que su utilidad no esté limitada, esto quiere decir que aunque como objetivo general se quiere que sirva como dispositivo de control, lo importante es el reconocimiento de los gestos de forma que el uso que se le dé luego en sus aplicaciones este abierto.

Haciendo uso de diferentes áreas de la electrónica se desarrolló un prototipo capaz de interpretar gestos, con un aprendizaje programable que permite su expansión tanto en instrucciones a reconocer o ejecutar (gestos), como en las situaciones que se puede utilizar gracias a una respuesta rápida y una implementación (solo se necesita ubicar los electrodos en el brazo) que es fácilmente utilizable por las aplicaciones que se deseen.

Generales:

Diseñar e implementar el prototipo de un dispositivo que permite reconocer un conjunto de acciones que el usuario puede realizar con la mano (gestos) y que estas puedan ser utilizadas para el control de aplicaciones en diferentes terminales.

Específicos:

Este dispositivo toma mediciones de señales eléctricas generadas por los músculos del antebrazo cuando estos se contraen y expanden (señales electromiográficas o EMG). A partir de la extracción de características de dichas señales y del análisis de las mismas, el dispositivo debe ser capaz de reconocer gestos realizados con la mano. El reconocimiento de los gestos se lleva a cabo por sistema de aprendizaje de máquinas, más específicamente una red neuronal artificial programable que a partir de un entrenamiento supervisado encuentre patrones que permitan detectar con éxito gestos realizados con la mano.

Para esto es necesario diseñar e implementar:

- Adquisición de las señales EMG en el antebrazo por medio de un arreglo electrodos.
- Diseño del circuito de acondicionamiento de las señales. Debe proveer un sistema robusto y que dé como resultado una señal compatible con diferentes microprocesadores.
- Procesar las señales. Incluye ordenar los datos (señales digitalizadas) para su análisis y permitir la consiguiente extracción de características.
- Establecer un conjunto de algoritmos para extracción de características que sean aplicables a este tipo de señales. Dichas características deben proveer suficiente información relevante sin abrumar la carga de trabajo realizada por el microprocesador.
- Entrenar una red neuronal por aprendizaje supervisado que sea capaz de reconocer los gestos.
- Recrear la red neuronal diseñada dentro del microprocesador. Esto permite que el sistema entero recaiga en un solo dispositivo.
- Asegurar que el proceso de adquisición, procesamiento y ejecución de red neuronal sea relativamente rápido. Esto responde a que el control de las aplicaciones no debe estar limitado por la velocidad de respuesta del dispositivo.
- Establecer una comunicación que sea compatible con las potenciales aplicaciones.
- Diseñar una aplicación demostrativa que sea controlada a partir de los gestos.

CAPITULO 1

MARCO TEORICO

1.1 Señales electromiográficas

El control de las fibras musculares descansa en unas unidades llamadas unidades motoras. Las unidades motoras generan potenciales de acción o MUAPs (motor unit action potential) para activar la generación de fuerza, por lo que al activarse un musculo se crean pequeñas corrientes a través de las fibras musculares.

Las señales electromiográficas o EMG son la superposición de los MUAPs que se generan en la activación de un musculo, como se observa en la figura 1.1 [4]. Ciertos tipos de análisis de señales EMG se basan en descomponer la señal en cada uno de los MUAPs que la forman.

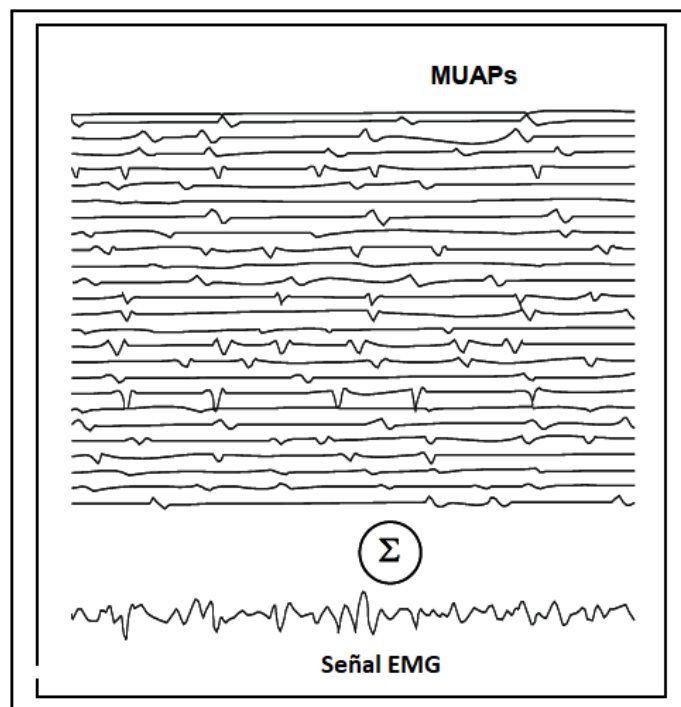


Figura 1.1 Composición de señal EMG [4].

Esta señal que no ha sido filtrada es denominada una señal EMG cruda. Esta señal es medida con ayuda de electrodos en la superficie de la piel o de forma invasiva con electrodos de cable fino.

1.2 Electrodos

Los electrodos son conductores eléctricos utilizados para hacer contacto con un elemento no metálico de un circuito. En el caso de mediciones EMG, estos se ponen en contacto con la zona de estudio para poder medir el potencial eléctrico.

Los electrodos se pueden dividir por la forma como realizan la conexión. Existen electrodos intrusivos o de punta fina que son una aguja insertada en la piel, la cual hace contacto directo con el musculo que está siendo estudiado. Estos permiten el monitoreo de un musculo en específico y proporcionan una medición más limpia. Por otro lado los electrodos superficiales son colocados en contacto con la piel, estos permiten una fácil aplicación y son aptos para situaciones que involucren movimiento del usuario, sin embargo en estos existe interferencia de los músculos adyacentes.

1.3 Aprendizaje de maquinas

El aprendizaje de máquinas es una rama de la inteligencia artificial, en la que se busca que las computadoras aprendan a partir de información suministrada para poder realizar un trabajo sin programación explícita [5]. Esto permite resolver problemas que por métodos convencionales presenta una alta dificultad, como búsqueda de patrones, clasificación, realizar estimaciones, entre otros. Por lo general se utilizan cuando no hay reglas claras de como separar la información, de forma que se pudiese armar un programa en unas cuantas líneas de código que hiciera el trabajo.

Con aprendizaje de máquinas existen diferentes métodos de aprendizaje. En el supervisado se enseña a partir de ejemplos, donde se da un conjunto de entradas y las salidas que

corresponden a cada caso. En el no supervisado no existe conocimiento previo de los datos y se busca describir una estructura para los datos que a priori parecen aleatorios.

Además de dar la solución fácil a los problemas difíciles, dan flexibilidad al permitir cambiar el programa a partir de nuevo entrenamiento y de esta forma modificar la tarea que realizan o mejorar su desempeño permitiendo una potencial evolución.

1.4 Red neuronal artificial espacio arriba

Las redes neuronales artificiales son un tipo de aprendizaje de máquinas, en el que se intenta simular el comportamiento de trabajo de las neuronas de un sistema nervioso [6]. En estas, un estímulo (entrada) es transmitido en una red de neuronas para dar la respuesta que corresponde. Cada conexión entre neuronas tiene un peso sináptico que representa la fuerza en la conexión. Este peso modifica la información que está siendo transmitida, de forma que cuando esta hace todo el recorrido, a la salida se tiene la respuesta correcta a dicho estímulo.

En una red neuronal se tienen una serie de capas por las que pasan los datos. Está la capa de entrada que son los estímulos; las capas ocultas contienen las neuronas y finalmente la capa de salida que podrían ser uno o varios elementos dependiendo de la cantidad de posibles respuestas. Cada una de las neuronas tiene múltiples conexiones de entrada y salida, y estas conexiones de forma similar al sistema nervioso, tienen pesos.

Las redes neuronales se pueden clasificar por la forma como se transmite la información entre capas. Entre las más comunes se tienen las prealimentadas y las recurrentes.

1.4.1 Redes neuronales prealimentadas

En una red prealimentada (“Feedforward”) cada neurona tiene una conexión por cada entrada y una conexión por cada neurona en la siguiente etapa. En este tipo de red la información fluye en un solo sentido como se observa en la figura 1.2.

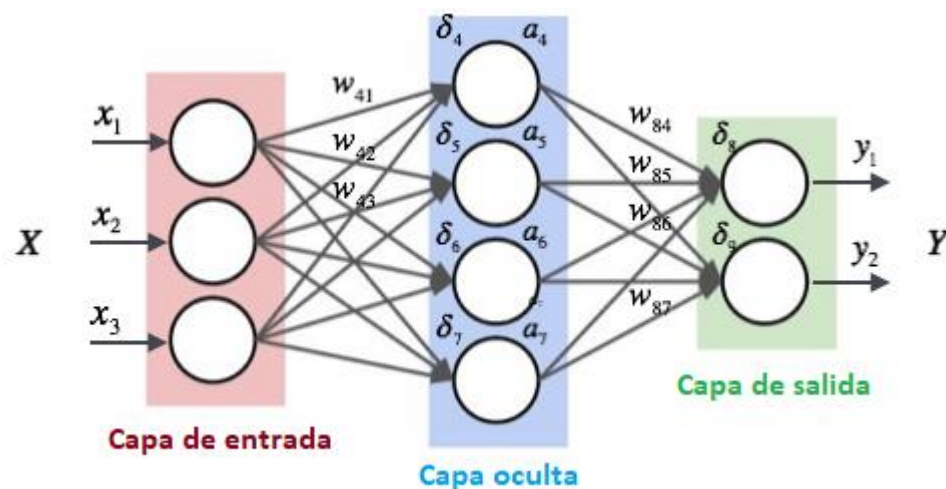


Figura 1.2 Red neuronal prealimentada.

1.4.2 Redes neuronales recurrente

En las redes recurrentes se tienen conexiones entre neuronas de la misma capa y capas anteriores, que hacen que la información fluya en múltiples sentidos como se observa en la figura 1.3. Aunque este tipo de red se asemeja más al sistema nervioso son más difíciles de entrenar y solo son realmente necesarias cuando una respuesta depende de datos presentes y pasados.

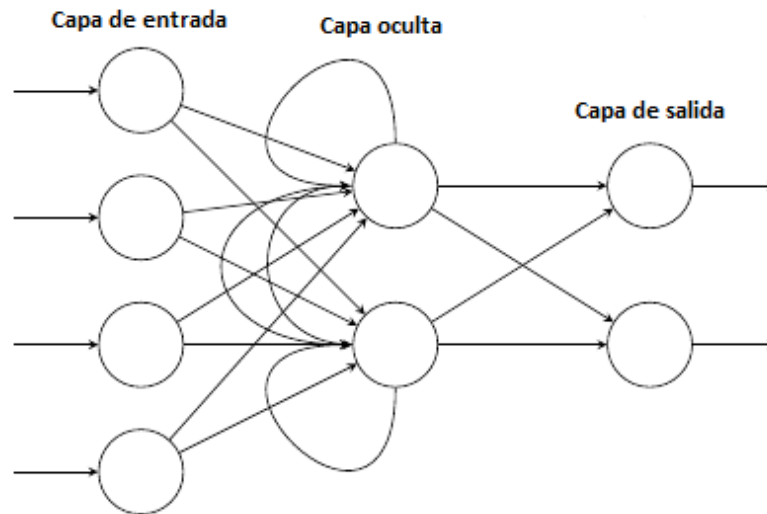


Figura 1.3 Red neuronal recurrente.

1.4.3 Tipos de neuronas espacio arriba

Las redes pueden tener diferentes tipos de neuronas. Los tipos vienen determinados por una función de transferencia, la cual cambia cómo se maneja la información en cada etapa [6]. La función de transferencia toma la información que viene de la capa anterior luego de haber sido modificada por los pesos, y la lleva a un espacio distinto con el que se puede, por ejemplo, tomar decisiones. Las más sencillas de entender son las lineales [6] [7], donde la neurona es simplemente la suma de todas las entradas por todos los pesos (Ec 1.1); otro tipo es por umbral binario [6], donde dependiendo del valor de todas las entradas, si este es mayor a un valor específico la neurona vale 1 de lo contrario vale 0 (Ec 1.2).

$$Y = b + \sum X_i \cdot W_i; \text{ donde } X_i = \text{entradas}, W_i = \text{pesos} \quad (1.1)$$

$$Z = b + \sum X_i \cdot W_i; Y = \begin{cases} 0 & \text{si } Z < \text{umbral} \\ 1 & \text{si } Z > \text{umbral} \end{cases} \quad (1.2)$$

En estas funciones está presente un elemento extra **b** llamado “bias” (tendencia). Esta es una constante que permite a la red modificar condiciones internas para la toma de decisiones. Se puede plantear como una neurona que siempre vale 1 y que solo tiene conexiones de salida como en la figura 1.4.

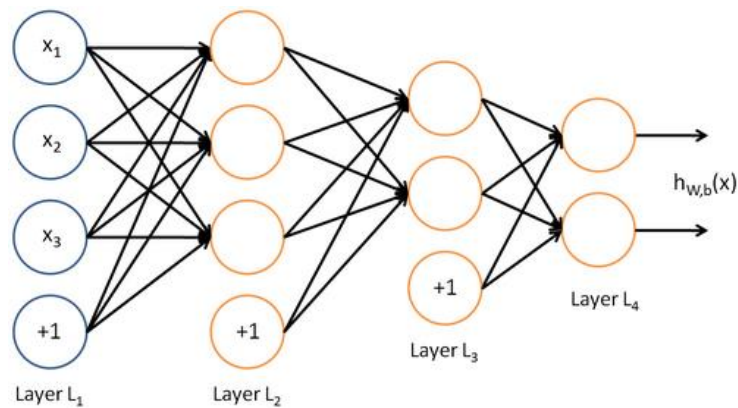


Figura 1.4 Diagrama de red neuronal con bias.

Las conexiones de estos elementos igualmente poseen pesos, sumando una constante b_i al resultado y permitiendo desplazar los resultados arrojados por las funciones de transferencia [6].

Para establecer la función de la red neuronal se necesita un algoritmo de aprendizaje. Su trabajo es dar el peso a cada conexión entre neuronas que resulte en la menor diferencia entre respuesta esperada y respuesta recibida. Para establecer el error existen diferentes métodos, uno comúnmente aplicado es la entropía cruzada.

1.4.1 Función de error: Entropía cruzada

Una de las formas de calcular la eficiencia de una red es con la función en entropía cruzada [8]. Entropía cruzada es utilizada para cuantificar la diferencia entre dos distribuciones probabilísticas, donde una es “verdadera” y la otra es la predicha [9]. Al ser aplicada en aprendizaje de maquinas la distribución “verdadera” es la que el algoritmo de aprendizaje está intentado igualar o predecir.

Si por ejemplo se tienen 3 posibles salidas y para una entrada específica la distribución verdadera es 0/1/0 esto representa que la probabilidad de pertenecer al segundo miembro es del 100%. Cuando la red neuronal está en ejecución un resultado podría ser 0,25/0,6/0,15, el cual representa su predicción. Para comparar estos resultados y determinar qué tan cerca se está de la distribución verdadera, se utiliza la ecuación 1.3.

$$H(p, q) = - \sum p(x) \cdot \log(q(x)); p: D. verdadera, q: D. predicha \quad (1.3)$$

Dada la forma de la función logarítmica (figura 1.5), al trabajar con valores entre 0 y 1 que son proporciones normalizadas, las grandes diferencias (valor predicho cercano a 0) son fuertemente penalizadas, mientras que predicciones cercanas al valor verdadero (cercanos a 1) tienen poca influencia en el error [10].

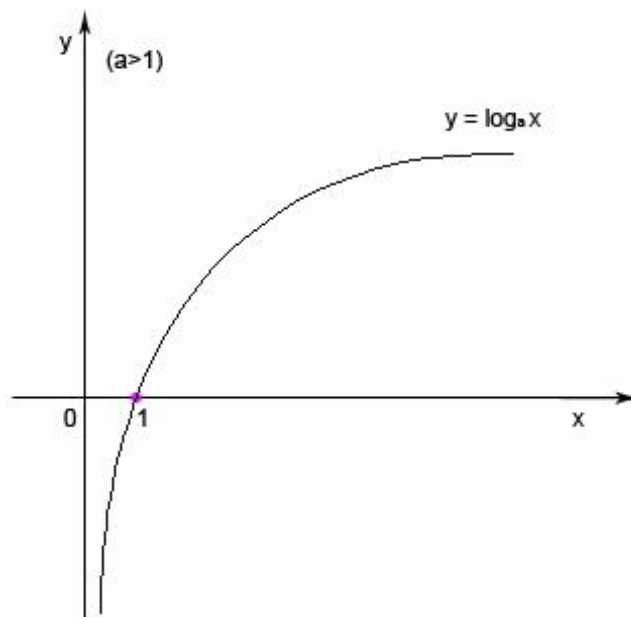


Figura 1.5 Función logarítmica.

1.4.2 Algoritmo de aprendizaje: Propagación hacia atrás

El algoritmo de aprendizaje es el que determina el comportamiento de la red a partir del cálculo de los pesos de cada conexión.

El algoritmo de propagación hacia atrás (“Backpropagation”), es principalmente utilizado en aprendizaje supervisado realizando un ciclo de propagación. Cuando se aplica una entrada a la red (un estímulo) este se propaga a través de las capas que componen la red hasta generar una salida. Dicha salida es comparada con el valor esperado, calculando el error para cada una. Dicho error es luego propagado hacia atrás, desde la capa de salida hacia las capas ocultas. Las neuronas reciben solo la fracción de la señal total de error, que corresponda al peso específico que tuvo cada neurona en el resultado final [11].

Este proceso causa que las neuronas se organicen y que cada una de ellas aprenda a reconocer las características del espacio de entrada total.

CAPITULO 2

DISEÑO CONCEPTUAL

En esta sección se describe cada uno de los subsistemas que comprenden el dispositivo de control por gestos, además del programa de demostración.

En el dispositivo se pueden distinguir 3 subsistemas. Estos trabajan en paralelo para permitir la detección en tiempo real y dependen uno del otro para el funcionamiento del dispositivo. Los subsistemas son: adquisición y acondicionamiento de la señal, el procesamiento de las mismas y la detección de los gestos. Un diagrama de dicha división se presenta en la figura 2.1.



Figura 2.1 Diagrama general del sistema.

2.1 Preprocesamiento de la señal

Este dispositivo toma las señales EMG que se miden en el antebrazo a partir de un conjunto de electrodos. Las características y las circunstancias en las que se encuentran estas señales crudas, hacen necesaria la existencia de una etapa que acondicione las señales, de forma

que estas sean admisibles y compatibles con el resto del sistema. Elementos como el tamaño de las señales medidas, el ruido existente en la medición, el generado por el movimiento o el potencialmente introducido por los componentes del sistema y las necesidades específicas del microcontrolador, son los elementos más importantes a ser considerados y solucionados en esta etapa.

El objetivo de este sistema es tomar las señales del arreglo de electrodos y llevarlas al punto de ser adquiridas por el microprocesador, como se describe en la figura 2.2.

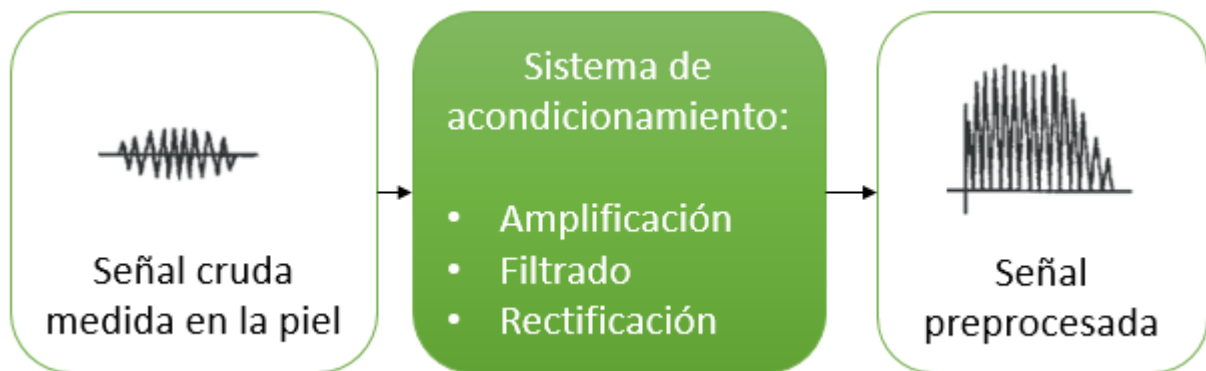


Figura 2.2 Diagrama del preprocesamiento de la señal

2.2 Procesamiento de las señales

Al tener la señal en un estado aceptable, es necesario llevarla al mundo digital para ser estudiada. La extracción de la información contenida en la señal EMG es la base para poder interpretar las acciones del usuario en el siguiente bloque del sistema.

Haciendo uso del ADC (conversor analógico digital) del microprocesador, se guarda una ventana temporal de las mediciones, es decir un segmento de la señal EMG como se observa en la figura 2.3. El tamaño de esta ventana responde a dos elementos principales, primero debe ser suficientemente representativo de lo que está sucediendo con la mano (tener suficientes datos) pero al mismo tiempo no puede ser tan grande que el procesamiento de la misma ralentice innecesariamente el proceso.

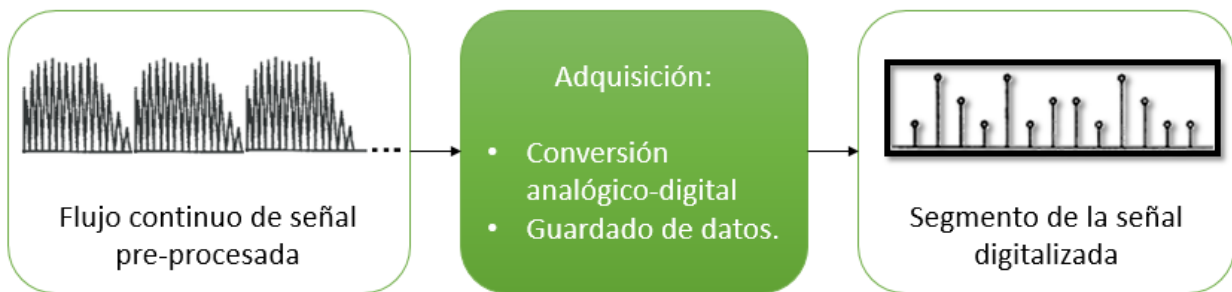


Figura 2.3 Diagrama de adquisicion de datos

Cada vez que se tiene uno de estos segmentos, se procede al procesamiento de los datos, que consiste en la extracción de características. Se tomó un conjunto de características que permiten obtener suficiente información relevante para poder identificar gestos. Con el conjunto de características, que consiste en un grupo de números (n números para n características), se alimenta el sistema de interpretación de datos. El proceso se describe en la figura 2.4.



Figura 2.4 Diagrama del procesamiento de la señal

2.3 Detección de gestos

La detección de los gestos se realiza en tiempo real dentro del mismo microprocesador. El encargado de interpretar los datos es una red neuronal artificial de detección de patrones diseñada en MATLAB.

El entrenamiento de la red neuronal se hace a partir de un conjunto de datos de entrenamiento. Este conjunto está comprendido por, un grupo de mediciones y las salidas esperadas para cada medición. Cada medición consiste en N características mientras que las salidas son M elementos uno por cada respuesta posible del sistema (M respuestas posibles corresponden a M gestos programados). Con estos dos elementos, la red busca patrones en la información de entrada de forma que se correspondan con la salida esperada. Las mediciones de entrenamiento son obtenidas a partir del sistema descrito en este trabajo.

Una vez entrenada la red y que se considera que el error es mínimo o aceptable, se pasa la red neuronal al microprocesador. Pasar la red neuronal, implica recrear la estructura entera de la misma, ya que no existe forma directa de llevar los datos de un lado a otro. Esto significó crear elementos que trabajen como las neuronas de la red, a partir de los pesos entregados por MATLAB.

Aunque hay elementos estáticos que no cambian entre entrenamientos (el flujo de los datos o las funciones de transferencia), los pesos de las neuronas si cambian. Por esta razón cada vez se entrena de nuevo la red, se deben actualizar los datos en el microprocesador ya que no existe un método automático para modificar los pesos. Esto implica ir al código y cambiar los valores en la variable que contiene todos los pesos.

La salida de la red es un conjunto de M números que representan el gesto o más específicamente la probabilidad que se esté ejecutando cada uno de los M gestos programados. El flujo general para la detección de gestos se tiene en la figura 2.5

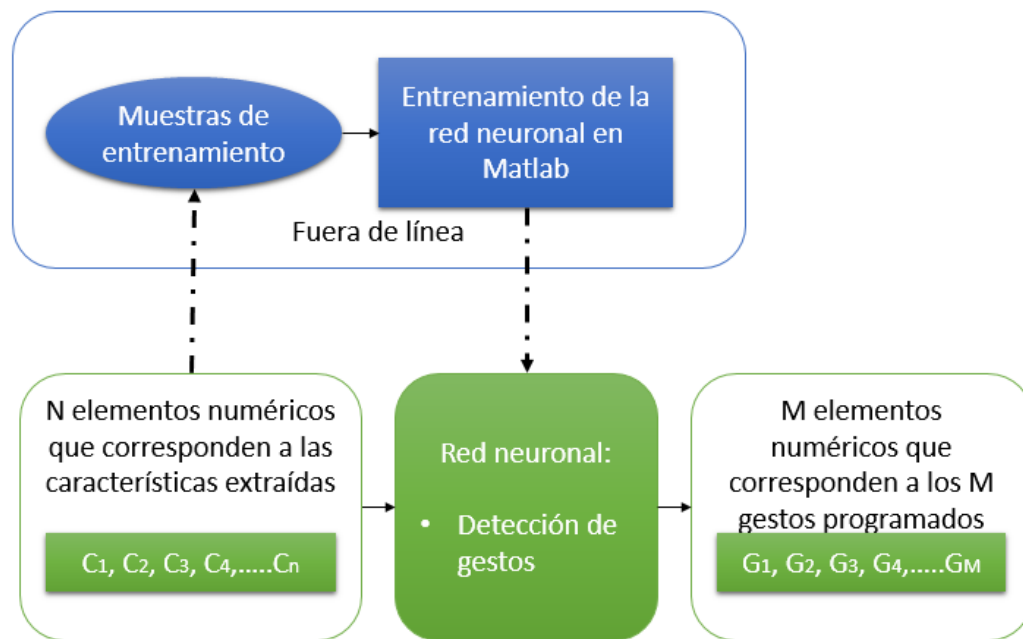


Figura 2.5 Diagrama de la detección de gestos

2.4 Programa de demostración

Finalmente se diseñó un programa en el motor de videojuegos Unity3D [12] con la única finalidad de demostrar que la detección de gestos no solo funciona y se ejecuta en tiempo real sino que es apta para el control de aplicaciones. Este programa permite diseñar con relativa rapidez y facilidad una aplicación constituida de un escenario donde un usuario interactúa con objetos. Usualmente la interacción es realizada con periféricos como teclado, un control de videojuegos e incluso la pantalla táctil de un dispositivo. Dado que Unity permite recibir instrucciones por comunicación serial, una aplicación desarrollada en este motor es un excelente ejemplo del reemplazo de controles convencionales por la opción que presenta el prototipo desarrollado.

Al programa desarrollado llega por comunicación serial un identificador del gesto detectado el cual es un número, y se le asocia a cada gesto una acción. El programa consiste en una representación 3D de una mano que realiza los mismos gestos que el usuario.

CAPITULO 3

SISTEMA DE ADQUISICION Y ACONDICIONAMIENTO

El primer sistema consiste en un circuito de acondicionamiento de las señales EMG. El diseño del circuito descrito en este capítulo sigue los lineamientos de los trabajos realizados por los ingenieros Guillermo Herrera y Génesis Urbina, quienes trabajaron en un sistema para el control de una prótesis de mano [13] y un sistema de control para un brazo robótico [14] respectivamente. Aunque el manejo de la información obtenida de las señales EMG y la aplicación final es distinta en los tres casos, la adquisición de las señales es similar.

El circuito de adquisición solventa una serie de potenciales problemas cuando se trabaja con este tipo de señales biológicas; además prepara la señal para ser completamente compatible con los requerimientos de la tarjeta de desarrollo utilizada (FRDM-K22F).

Para el acondicionamiento de la señal se diseñó un circuito que implementa diferentes etapas de amplificación, filtrado y finalmente de rectificación de la señal. El objetivo es dejar la señal en un estado utilizable por el ADC del microprocesador.

El circuito consta de los electrodos, etapa de preamplificación, filtro pasa alto, amplificación, una etapa de rectificación y filtro pasa bajo, además se describe el circuito de alimentación implementado.

Es necesario mencionar que el orden de cada etapa entre las secciones 3.1 y 3.5 no es el utilizado en el circuito entero, las secciones están ordenadas por factores de diseño. El diagrama exacto del circuito de acondicionamiento y el orden de cada circuito se muestra en la sección 3.6.

3.1 Electrodos

3.1.1 Diseño

En este proyecto fue necesario utilizar electrodos superficiales dado que electrodos intrusivos no son compatibles con el concepto del proyecto. La calidad de la medición con este tipo de electrodos depende de la impedancia electrodo-piel, la cual entre más baja mejor. Existen

electrodos no polarizables de Ag/AgCl (plata/cloruro de plata) como los de la figura 4.1, los cuales son los más utilizados en mediciones médicas [15] como electrocardiogramas, electromiografías o electroencefalogramas. Este tipo de electrodos poseen una capa de gel electrolítico que facilita las reacciones electroquímicas que suceden entre la piel y el electrodo. Estas reacciones de oxidación-reducción determinan la relación señal a ruido ya que una alta impedancia en la interface electrodo-piel se traduce en una baja movilidad en el intercambio ion-electrón [16].

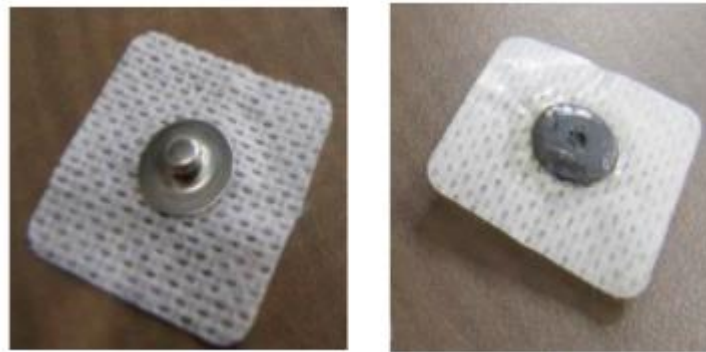


Figura 3.1 Ejemplo de electrodo de Ag/AgCl [16].

Aunque los electrodos utilizados en este trabajo son de Ag/AgCl, la implementación futura de este dispositivo tendría que hacer el cambio a electrodos secos como los de acero inoxidable o a electrodos orbitales similares a los de las figuras 3.2 y 3.3 respectivamente, los cuales son una placa de metal que se pone en contacto con la piel. Utilizar este tipo de electrodo aunque aumentaría la impedancia, elimina la necesidad de un cambio constante de electrodos y facilitaría el uso del dispositivo.



Figura 3.2 Ejemplo de electrodo de acero inoxidable [16].



Figura 3.3 Ejemplo de electrodo orbital [16].

Una medición a partir de electrodos, sea cual sea el caso, requiere por lo menos 3 de estos, un par para realizar la medición diferencial y un tercero de referencia. En el caso de este proyecto se miden de forma simultánea dos sectores en el antebrazo, lo que implica 5 electrodos, dos pares para cada medición y el de referencia ubicada en el codo. La ubicación de este último responde a que se recomienda ubicar el electrodo de referencia en una zona no cercana al musculo estudiado y zonas cercanas a huesos son especialmente útiles [4].

Debido a limitaciones en los componentes disponibles, dos mediciones simultáneas es el máximo en esta implementación, ya que cada par de electrodos, es decir cada señal EMG medida, requiere un circuito de acondicionamiento entero.

Cabe mencionar que el código implementado en el microprocesador hace que el escalar el proceso a más señales sea relativamente sencillo.

3.1.2 Implementación

Los electrodos utilizados son electrodos de Ag/AgCl como los de la figura 3.4, fabricados por Covidien [17]. Estos son unos electrodos desechables que tienen una esponja de polietileno en el borde con un agente adhesivo que permite su fácil conexión a la piel. Estos poseen un hidrogel sobre el electrodo que ayuda a la conductividad, mientras que el conector externo (que va hacia el cable) es de acero inoxidable.



Figura 3.4 Electrodos de Ag/AgCl

Para hacer la conexión de estos al sistema de acondicionamiento, se utilizó unos cables que son usualmente utilizados para electroterapia como los de la figura 3.5. Estos son unos cables multifilares, a los cuales se les elimino el conector para poder realizar una conexión directa al circuito en la placa de pruebas.

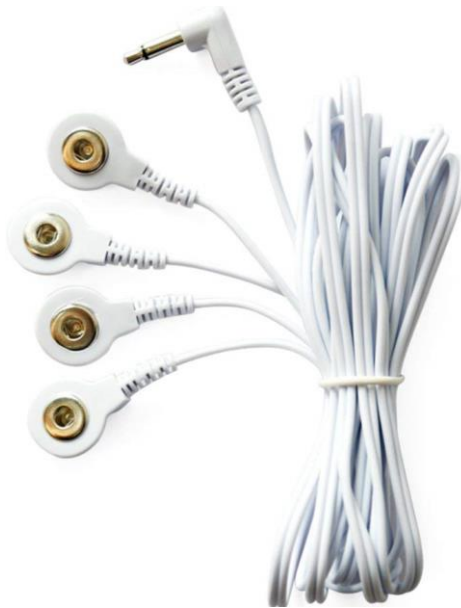


Figura 3.5 Cables de los electrodos

A partir de electrodos superficiales se tiene acceso al músculo flexor cubital del carpo y al músculo braquiorradial en el antebrazo; observables en la figura 3.6.



Figura 3.6 Músculos medidos.

La posición de los electrodos se pueden observar en las figuras 3.7, 3.8 y 3.9.



Figura 3.7 Medición del músculo braquiorradial.



Figura 3.8 Medición del músculo flexor cubital del carpo.



Figura 3.9 Electrodo de referencia en el codo.

La longitud de los cables es de aproximadamente 65cm. Estos cables deberían ser muchos más cortos, de hecho idealmente se tendría una conexión prácticamente directa entre los electrodos y el circuito de acondicionamiento. Sin embargo dado el montaje del circuito en la placa de pruebas, hacer los cables más cortos haría extremadamente difícil hacer las pruebas.

3.2 Amplificación

3.2.1 Diseño

Las señales EMG son señales pequeñas que no se prestan para ser utilizadas de forma directa. Estas señales varían en el rango de los $5\mu\text{V}$ hasta 5mV pico [4], por esta razón se implementó una serie de etapas de amplificación que llevan la señal a una ventana de voltaje más acorde a la aplicación en mano. En este caso, la tarjeta de desarrollo permite trabajar (de forma nativa) con señales de hasta $3,3\text{V}$.

La amplificación fue dividida en dos etapas. La primera etapa de preamplificación es la que está más cercana a la señal EMG, recibiendo directamente lo que miden los electrodos y una ubicada más adelante que lleva la señal a un valor cercano a los 3.3V objetivo.

En la pre-amplificación existen una serie de factores importantes a tener en cuenta:

- Un alto factor de rechazo al modo común:

Al utilizar un amplificador diferencial para el arreglo de electrodos, se están eliminando las señales comunes a ambos. Las señales que existen en toda la superficie del ser humano están correlacionadas, por lo que se detectan en ambos puntos de medición; entre estas se encuentra las generadas por las fuentes de voltaje, por señales electromagnéticas de dispositivos que nos rodean e incluso señales EMG de músculos distantes que no están siendo estudiados. El factor de rechazo al modo común o CMRR (common mode rejection ratio) por sus siglas en inglés, sirve como índice de la atenuación a estas señales comunes. Un valor recomendado en adquisición de señales EMG es un CMRR > 80dB [15] [18].

- Una alta impedancia de entrada

La impedancia en la fuente (interfaz piel-electrodo) para electrodos con gel está por debajo de los 50K Ω [15]. Para poder realizar una buena medición, la resistencia en la entrada del circuito debe ser considerablemente mayor a la impedancia en la fuente. De lo contrario la señal será atenuada y distorsionada debido al efecto de carga en la entrada. La recomendación para impedancia de entrada en mediciones EMG es que esta debe ser mayor a 100M Ω para electrodos de gel [18] mientras que para electrodos secos, se necesita en el orden de los Giga ohmios.

- Corta distancia a la fuente (electrodos)

Una de las desventajas de tener una alta impedancia en la entrada es que el ruido de la fuente de potencia, el ruido de radio frecuencias y el ruido por el movimiento de los equipos son introducidos en el cableado a causa del acoplamiento capacitivo. Por tanto entre más largo es el

cableado entre el electrodo y el circuito, la capacitancia aumenta y con esto el ruido, resultando en una relación señal a ruido reducida.

- Fuerte supresión de la señal DC

Por último es importante que el circuito de pre-amplificación elimine todo componente DC medido. Existen componentes DC generados por la impedancia de la piel y la reacción química entre la piel y los electrodos. Cualquier diferencia DC en los potenciales medidos es amplificada, lo que puede causar inestabilidad y hasta saturación.

Los problemas de un alto CMRR y una alta impedancia de entrada, fueron solucionados con el componente seleccionado; un amplificador de instrumentación descrito en la siguiente sección. En cuanto a eliminar el componente DC se implementó un circuito de filtro pasa bajo acoplado al amplificador de instrumentación, que controla su referencia. Al final se pasa de la débil señal medida por los electrodos a una señal en el orden de los cientos de milivoltios. El diagrama de esta etapa se puede observar en la figura 3.10.



Figura 3.10 diagrama de pre-amplificación de la señal

Ya que en la etapa de preamplificación el principal objetivo es adquirir la señal medida por los electrodos y responder a los problemas de dicha adquisición, existe una segunda etapa de amplificación más sencilla que lleva la señal del orden de los cientos de milivoltios a un valor más cercano a los límites del microcontrolador de 3,3V.

3.2.2 Circuito de preamplificación

Los requerimientos iniciales (refiriéndonos a primera etapa en la adquisición), hacen que un amplificador de instrumentación sea una excelente opción a implementar. Estos dispositivos presentan una alta impedancia de entrada junto con un alto CMRR, razón por la cual son muy utilizados cuando se trabaja con señales débiles.

Consta de un trio de amplificadores operacionales, que restan dos señales y la amplifican. En este caso se mide la diferencia de potencial entre el par de electrodos y luego es amplificada. En la figura 3.11 se puede observar el diagrama del amplificador de instrumentación INA121 [19] utilizado.

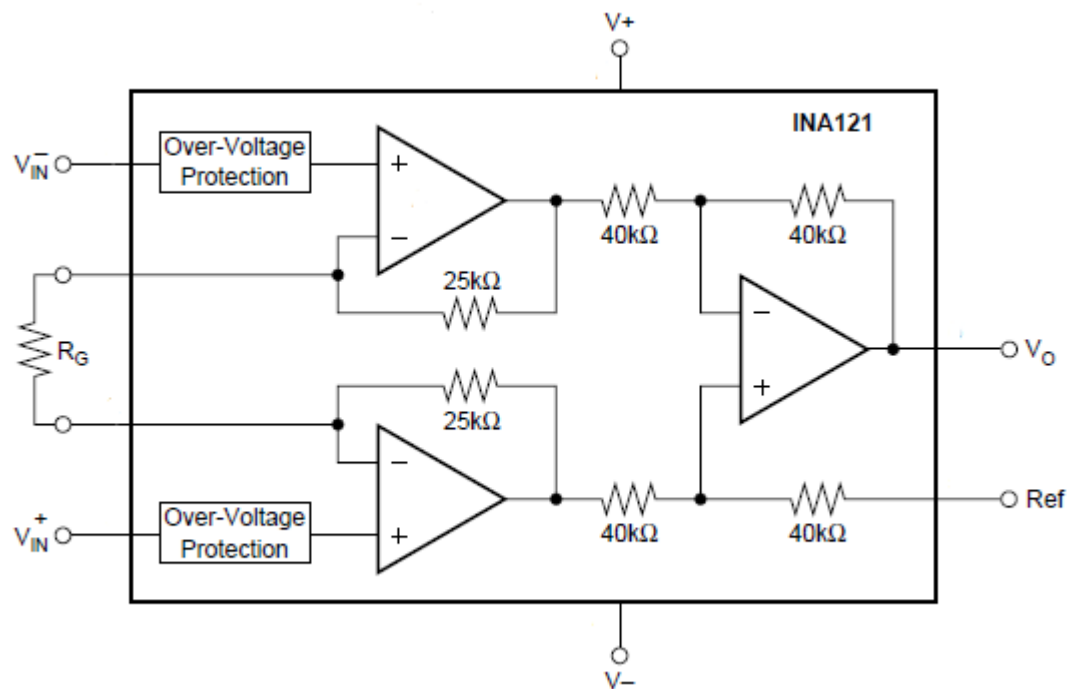


Figura 3.11 Amplificador de instrumentación INA121 [19].

Este componente puede ser alimentado de $\pm 2,25\text{V}$ hasta $\pm 18\text{V}$, posee ganancia variable de hasta 10000V/V , presenta un CMRR de 106dB (a una ganancia de 100V/V) y una protección de hasta $\pm 40\text{V}$.

La ganancia en este componente se controla con una resistencia externa, y su valor viene dado por la ecuación 3.1.

$$G = 1 + \frac{50K\Omega}{R_g} \quad (3.1)$$

En esta primera etapa de amplificación, se decidió utilizar una ganancia de 100V/V, lo cual significa que la señal a la salida ronda los cientos de mili-voltios.

Partiendo de la ganancia objetivo se obtiene que la resistencia R_g a utilizar es:

$$G = 100 \Rightarrow R_g = \frac{50K\Omega}{G - 1} = 505\Omega \quad (3.2)$$

Que llevándolo a un valor comercial es una resistencia de 510Ω.

A este elemento de amplificación se le acoplo un circuito que controla su referencia permitiendo filtrar cualquier componente DC que pudiese existir a la salida. Esto se realizó armando un filtro pasa bajo de primer orden a la salida del INA con el que se establece un lazo de retroalimentación hacia su referencia como se muestra en la figura 3.12. Al pasar estos componentes como referencia en esencia se crea un filtro pasa alto.

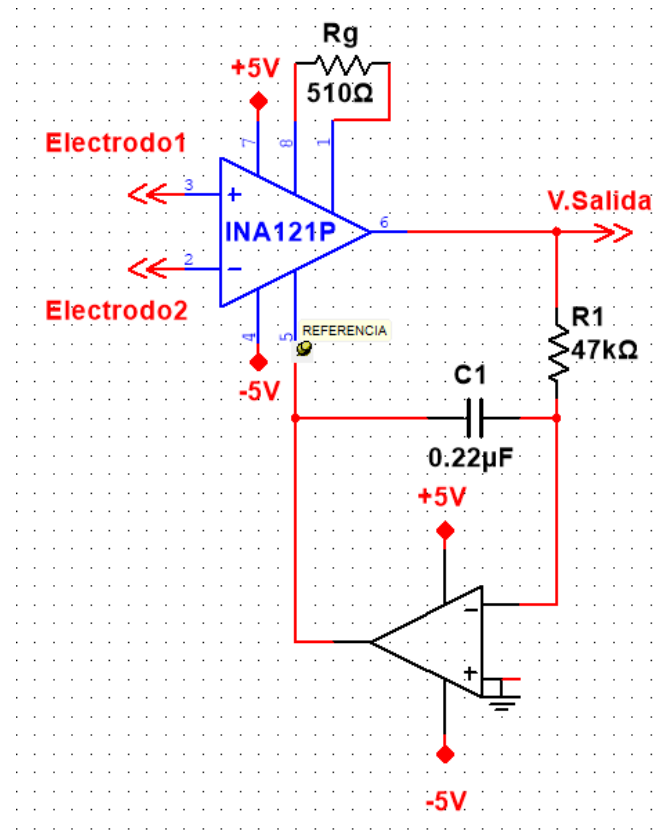


Figura 3.12 Amplificador de instrumentación con retroalimentación implementado.

Con los valores del capacitor C_1 y la resistencia R_1 se controla la frecuencia de corte del filtro según la ecuación 3.3.

$$F_{3dB} = \frac{1}{2\pi R_1 C_1} \quad (3.3)$$

En la sección 3.3 se describe la banda de frecuencias con las que se trabaja en este proyecto. Dejando la explicación para dicha sección, en este trabajo se utilizan las frecuencias mayores a 20Hz. Ya que se buscaba eliminar los componentes DC y solo se utilizan las frecuencias mayores a 20Hz, la frecuencia de corte objetivo para este filtro es una por debajo de dicho valor. Con esto en mente y tanteando diferentes combinaciones de valores para los componentes disponibles, se estableció como $R_1=47K\Omega$ y $C_1=0,22\mu f$ quedando como resultado:

$$F_{3dB} = \frac{1}{2\pi \cdot 47K\Omega \cdot 0,22\mu f} \approx 15,4Hz \quad (3.4)$$

El diagrama del circuito implementado se ve en la figura 3.12.

3.2.3 Segunda etapa de amplificación

Para la etapa final de amplificación se implementó un simple amplificador no inversor que consta de un operacional y un par de resistencias como se observa en la figura 3.13. Con esta etapa se lleva la señal de los cientos de milivoltios al orden de unos pocos voltios para maximizar el rango aceptado por el ADC del microprocesador.

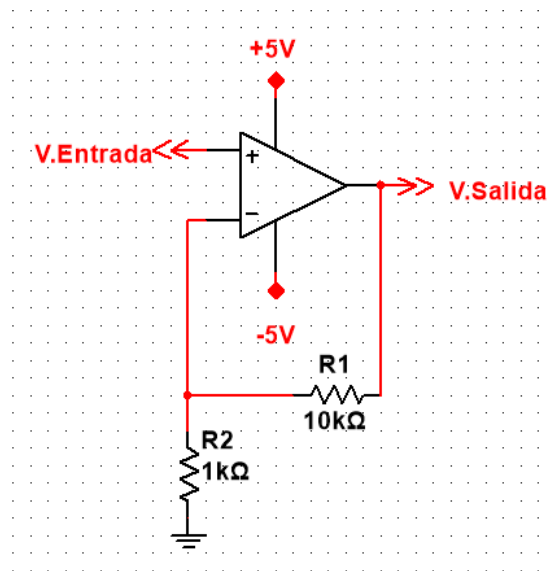


Figura 3.13 Amplificador no inversor implementado.

Con la ecuación que describe al circuito se calculan los valores de R_1 y R_2 que nos entreguen la ganancia deseada, la cual fue establecida en $G=11V/V$.

$$G = 1 + \frac{R_1}{R_2} = 11 \xrightarrow{R_2=1K\Omega} R_1 = 10K\Omega \quad (3.5)$$

Con esta configuración se obtiene una amplificación total (ambos circuitos de amplificación) de aproximadamente 1100V/V.

3.3 Filtros

3.3.1 Diseño

El sistema requiere de filtros que eliminen todo lo que se encuentre fuera de la banda de trabajo. La mayor parte del componente energético de las señales EMG se encuentra en el rango de los 20Hz a los 500Hz [18] por lo que el resto es prescindible. Con esto en mente el uso de filtros es tanto útil como necesario.

Una recomendación para el filtrado de señales EMG es que la pendiente en la frecuencia de corte de los filtros sea de al menos 12dB/octava o -40dB/década [18].

Dado que implementar un filtro pasivo de alto orden podría presentar problemas de acoplamiento de impedancias, resulta más recomendable implementar un filtro activo. Para determinar qué tipo de filtro era el correcto fue necesaria una comparación entre diferentes filtros [20].

Chebyshev: tienen una caída pronunciada en la frecuencia de corte, sin embargo presentan oscilaciones (ripple) en la banda de paso, lo cual se traduce en distorsiones.

Bessel: tienen un comportamiento suave en la banda de paso pero una transición en la frecuencia de corte muy larga. Poseen la mejor respuesta a cambios de fase por lo que responden bien a cambios en el nivel de las señales.

Elípticos: presentan una buena pendiente pero con oscilaciones tanto en la banda de paso como en los componentes fuera de la banda.

Butterworth: tienen un comportamiento plano en la banda de paso pero con una pendiente menor a los que se obtienen con un filtro Chebyshev del mismo orden.

En la figura 3.14 se puede observar la diferencia de comportamiento para cada uno de estos tipos de filtros.

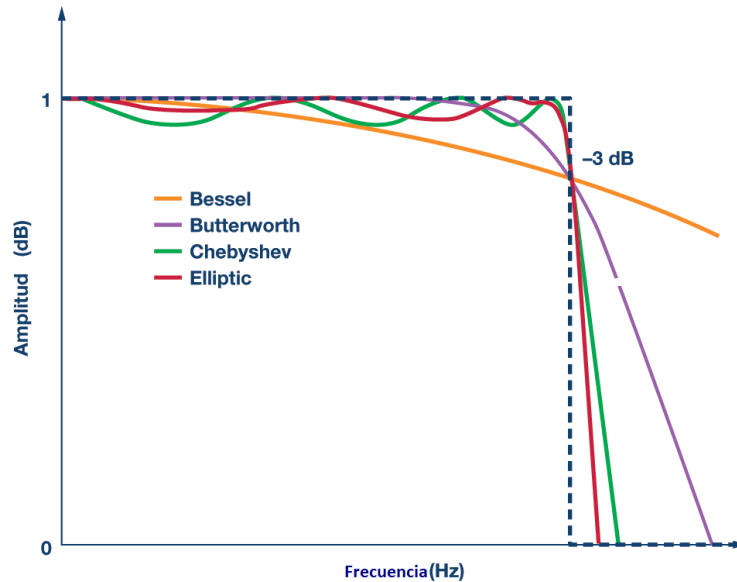


Figura 3.14 Comparación de filtros

Dado que lo más importante en este proyecto es mantener la integridad de la señal, la respuesta plana de un filtro Butterworth lo hace la opción correcta. La relación de cada polo con la pendiente en un filtro Butterworth es de 20dB/década; esto significa que los filtros implementados debían ser de al menos 2do orden para cumplir con las recomendaciones. Para un filtro Butterworth de segundo orden el factor de calidad es:

$$Q = \frac{1}{\sqrt{2}} = 0,707 \quad (3.6)$$

Aunque en la etapa de pre-amplificación se implementó un filtro pasa alto para eliminar los componentes DC que se puedan filtrar al sistema, es necesario un filtro pasa alto específico para eliminar todos los componentes de baja frecuencia que cumpla con los parámetros antes descritos.

Con estos factores en mente, las etapas de filtrado implementadas responden a: filtros Butterworth ($Q=0.707$), pasar la banda de frecuencias de 20Hz a 500Hz y presentar una pendiente de al menos -40dB/década (ser de al menos 2do orden).

3.3.2 Filtro pasa alto Switched capacitor

Esta etapa debe eliminar todo componente frecuencial por debajo de los 20Hz. Los filtros con topología Sallen Key son filtros activos relativamente sencillos, los cuales dependen solo de un arreglo de componentes pasivos (resistencias y capacitores) para establecer su frecuencia de corte y funcionar [21]. En la figura 3.15 se puede observar el diagrama del filtro pasa alto Sallen Key utilizado.

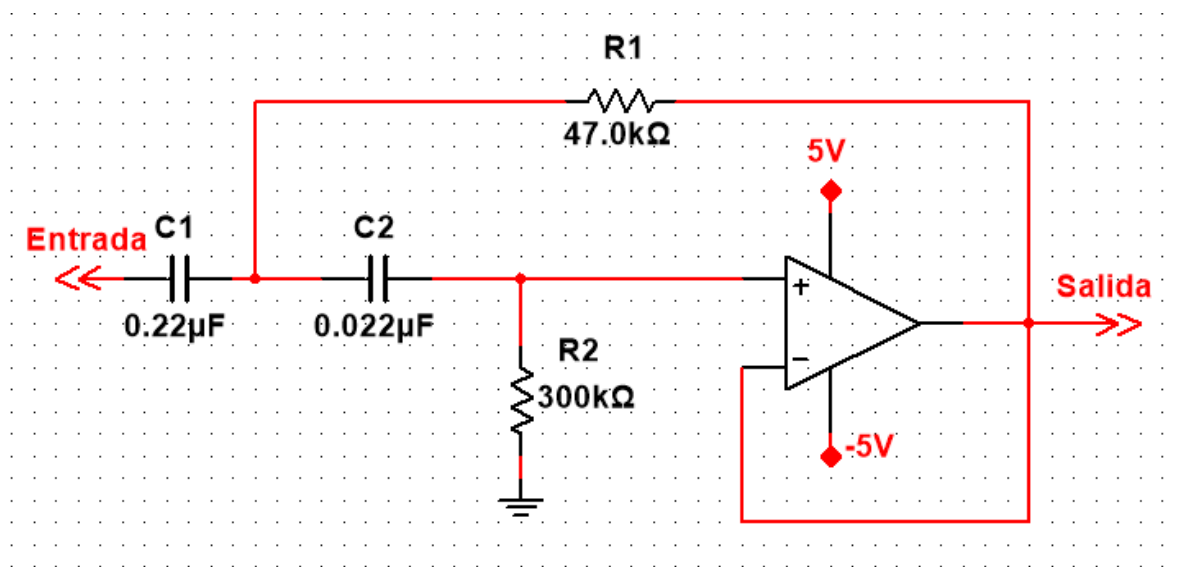


Figura 3.15 Filtro pasa alto Sallen Key implementado.

Las ecuaciones con las cuales se describe este tipo de filtro son:

$$F_c = \frac{1}{2\pi\sqrt{R_1 R_2 C_1 C_2}} \quad (3.7)$$

$$Q = \frac{\sqrt{R_1 R_2 C_1 C_2}}{R_1 (C_2 + C_1)} \quad (3.8)$$

Con $F_c=20\text{Hz}$ y $Q=\frac{1}{\sqrt{2}}$, se tienen 2 funciones con parámetros conocidos y 4 incógnitas, las cuales pueden ser controladas. Como procedimiento de diseño común, se tiende a establecer una proporción entre los valores de las resistencias y otro entre los valores de los capacitores [21].

$$R_1 = mR = mR_2; \quad C_1 = nC = nC_2 \quad (3.9)$$

$$F_c = \frac{1}{2\pi RC\sqrt{mn}} = 20\text{Hz} \quad (3.10)$$

$$Q = \frac{\sqrt{mn}}{m(n+1)} = \frac{1}{\sqrt{2}} \quad (3.11)$$

Por simplicidad se establece $n=10$ y se extrae m de la segunda expresión.

$$\sqrt{2mn} = m(1+n) \quad (3.12)$$

$$\Rightarrow \sqrt{20mn} = m(11) \quad (3.13)$$

$$\Rightarrow 20m = m^2 121 \quad (3.14)$$

$$\Rightarrow m = \frac{121}{20} = 0,165289 \quad (3.15)$$

Con los valores de m y n , se calcula R y C de la relación 3.10.

$$RC = \frac{1}{\sqrt{mn}(2\pi \cdot 20\text{Hz})} = 6,189679\text{m}\Omega\text{f} \quad (3.16)$$

Finalmente para despejar el conjunto de componentes se establece el valor de C_2 en $0,022\mu\text{F}$.

$$C_2 = C = 0,022\mu\text{F} \Rightarrow C_1 = nC = 0,22\mu\text{F} \quad (3.17)$$

$$R_2 = \frac{6,189679m\Omega F}{0,022\mu F} \approx 281,349K\Omega \quad (3.18)$$

$$R_1 = mR \approx 46,503k\Omega \quad (3.19)$$

Buscando valores comerciales para estos componentes, se tiene que R_1 y R_2 son $47K\Omega$ y $300K\Omega$ respectivamente. Con estos nuevos valores se recalculo la frecuencia de corte y el factor de calidad resultando: $F_c=19,27Hz$; $Q=0,7263$.

El diagrama del circuito se encuentra en la figura 3.15.

3.3.3 Filtro pasa bajo

Con este filtro se busca eliminar los componentes frecuenciales mayores a $500Hz$. Aunque de forma similar al filtro pasa alto, se podría implementar otro filtro Sallen Key, era preferible utilizar un filtro integrado que tuviese una respuesta aún más efectiva, aumentando la pendiente, eliminando todo el ruido de alta frecuencia y minimizando errores al disminuir el número de componentes que habrían sido necesarios.

Para esto se utilizó el integrado MAX7480 [22], un filtro Butterworth pasa bajo de 8vo orden. Este filtro funciona con una alimentación máxima de hasta $5V$, razón por la cual la rectificación de la señal se realiza previa a este filtro en el circuito final; de lo contrario se tendría que implementar un circuito de alimentación solo para este componente ($\pm 2,5V$) y realizar el filtrado antes de la segunda etapa de amplificación.

La frecuencia de corte puede ser establecida desde $1Hz$ hasta $2KHz$. El control de dicha frecuencia se puede realizar por dos métodos; el primero es utilizando el reloj interno del componente, conectando un capacitor con cuyo valor se controla la frecuencia, la segunda opción es a partir de un reloj externo.

Por descripción de la hoja de datos, el control de la frecuencia a partir de un reloj externo es más exacta, por lo que se decidió controlar este filtro con una señal de control generada con el microcontrolador. En este método la frecuencia de corte cumple con la ecuación 3.20. Dada la frecuencia de corte requerida de 500Hz, se tiene:

$$F_c = \frac{F_{clk}}{100} \xrightarrow{F_c=500Hz} F_{clk} = 50KHz \quad (3.20)$$

El componente requiere que la señal de control tenga un ciclo de trabajo entre 40-60% y un voltaje que este entre 0 y el voltaje de alimentación, en este caso 5V. Para generar esta señal se utilizó el FTM (flex time module) [23] del microcontrolador, el cual permite generar una señal cuadrada, con ciclo de trabajo variable, con una frecuencia controlable y un voltaje de 3,3V.

En esta implementación, la fuente de reloj del FTM es el reloj de BUS del microcontrolador, establecido en aproximadamente 20MHz. Para la generación de la señal de control se trabaja en modo de comparación de la salida (output compare), donde se establece un contador que al llegar al número objetivo cambia la polaridad de la señal. Ya que se desea una señal con una frecuencia de 50KHz y la frecuencia que se establece en el FTM es la de cambio, significa que esta última debe ser el doble para quedar con una señal cuadrada con un ciclo de trabajo del 50%. En la figura 3.16 se observa un ejemplo de la forma de la señal.

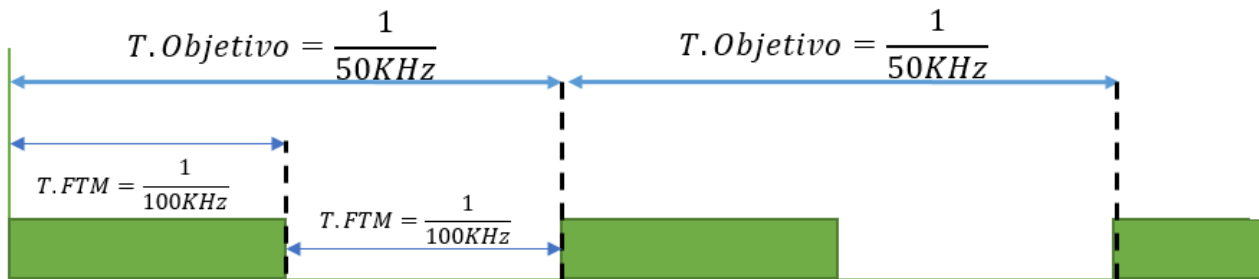


Figura 3.16 Señal de control (FTM).

Siguiendo los lineamientos de la hoja de datos, el componente queda conectado como se observa en la figura 3.17. Teniendo en el terminal CLK la conexión a la señal de control proveniente del microcontrolador.

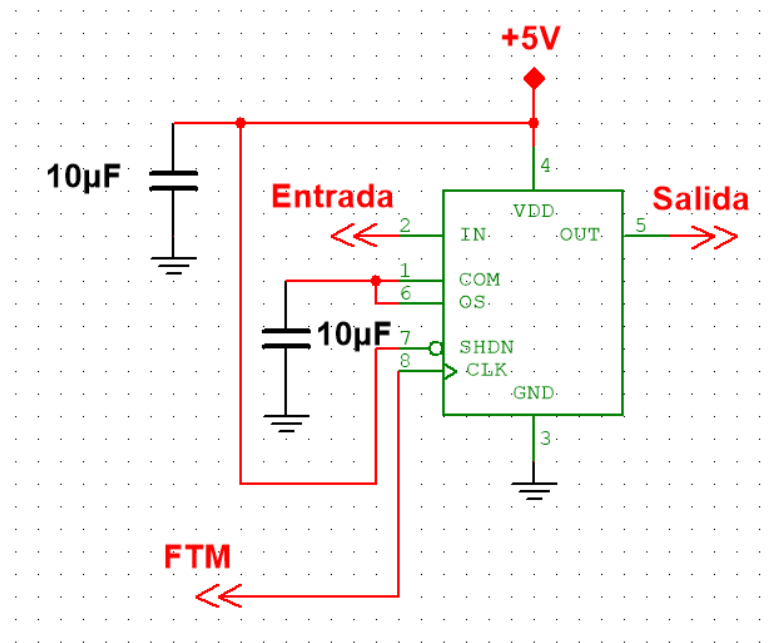


Figura 3.17 Filtro pasa bajo MAX7480.

Este filtro también cumple con el objetivo de evitar un posible solapamiento o aliasing al realizar el muestreo de las señales. En la sección 4.1 del presente documento se describe que la frecuencia de muestreo en el microprocesador fue establecida en aproximadamente 12 KHz; esto significa que la frecuencia de corte máxima aceptable para evitar el solapamiento es de 6KHz. Dada esta gran diferencia con la frecuencia de corte establecida, significa que el orden del filtro pasa bajo podría haber sido menor y aun así no sería un problema.

3.4 Rectificación

La señal diferencial que se obtiene de los músculos posee componentes negativos y positivos. Aunque esto no es un problema propiamente dicho, poseer una señal de una sola polaridad permite tener valores de media distintos a 0 y hace que la señal sea más fácil de manejar desde el punto de vista de la digitalización. Para esto se utilizó el rectificador descrito

en la publicación “TI Precision Designs: Verified Design Precision Full-Wave Rectifier” [24], cuyo diagrama se puede observar en la figura 3.18.

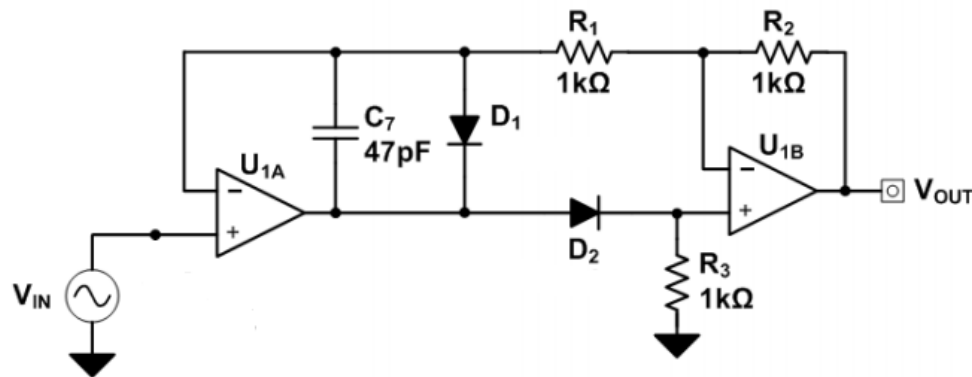


Figura 3.18 Circuito de rectificación de onda completa [24].

En este rectificador los operacionales U_{1a} y U_{1B} polarizan los diodos D_1 y D_2 , para cambiar el camino de la señal a partir de la polaridad de la señal de entrada. Según la polaridad de la señal el circuito se comporta como en la figura 3.19 o la figura 3.20 para positivo y negativo respectivamente.

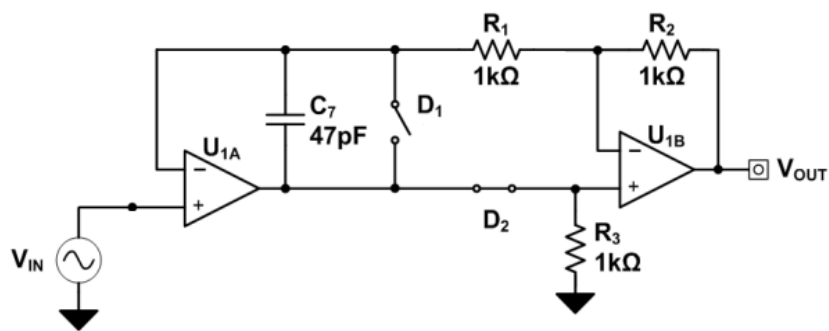


Figura 3.19 Diagrama del rectificador para polarización positiva [24].

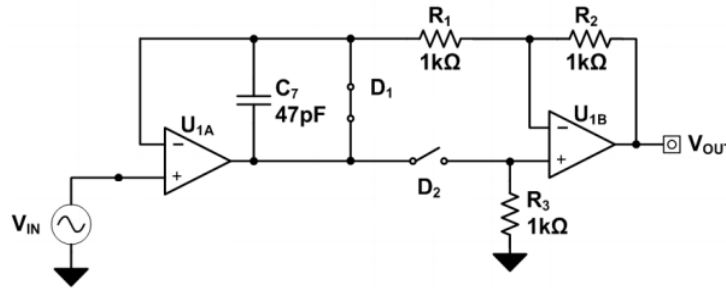


Figura 3.20 Diagrama del rectificador para polarización negativa [24].

Cuando la señal es positiva, D_1 está en polarización inversa a diferencia de D_2 que funcionan en directo, por lo que estos se comportan como un abierto y un corto respectivamente. U_{1A} controla el terminal no inversor de U_{1B} , además no se tiene corriente circulando hacia el terminal inversor de U_{1A} por tanto las resistencias R_1 y R_2 tampoco tienen corriente circulando por ellas. Esto resulta en que U_{1B} funcione como un simple seguidor de voltaje por lo que significa que U_{1A} también.

En polarización negativa, D_1 está de directo mientras que D_2 es un abierto, esto nos deja con U_{1A} funcionando como un seguidor, mientras que U_{1B} es un amplificador inversor que tiene como ganancia la expresión 3.20. Esto nos deja con la señal positiva.

$$G = -\frac{R_2}{R_1} = -1 \frac{V}{V} \quad (3.21)$$

El capacitor C_7 existe para dar un camino de realimentación a las altas frecuencias que ayuda a estabilizar la salida del primer operacional; este no puede ser de un valor muy alto ya que distorsionaría los bordes donde la señal cambia de polaridad

3.5 Alimentación

El sistema de alimentación está diseñado para dar energía a todos los elementos del sistema de acondicionamiento. Todos los componentes se eligieron de forma que necesitaran ser alimentados por fuentes de $\pm 5V$ por lo que un solo sistema de alimentación es suficiente.

Como elementos reguladores de voltaje se utilizaron los integrados L7805 [25] (figura 3.21) y L7905 [26] (figura 3.22) que corresponden al regulador positivo y negativo

respectivamente. Ambos tienen una salida máxima de 1,5A, protección de cortocircuito y recalentamiento, y poseen una variación máxima en la salida del 0,1V. El diagrama del circuito se puede observar en la figura 3.23.

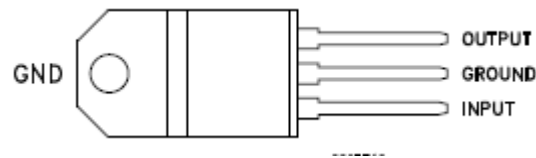


Figura 3.21 Regulador positivo L7805 [25].

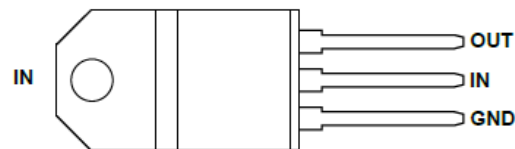


Figura 3.22 Regulador negativo L7905 [26].

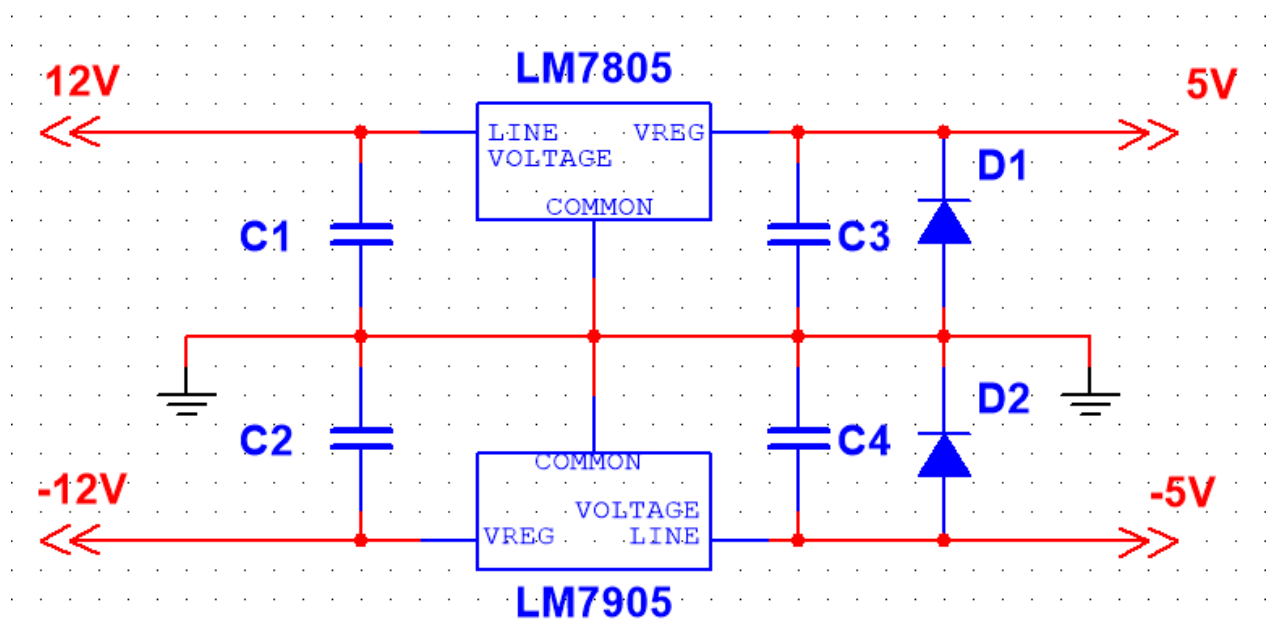


Figura 3.23 Circuito de alimentación.

En esta implementación, la fuente de voltaje es una fuente de computadora, en la cual se hace uso de las conexiones de $\pm 12V$ que posee como salidas.

Los capacitores de desvío y los diodos son una recomendación de la hoja de datos de los componentes [25] [26]. Los capacitores a la entrada (C_i) son útiles cuando hay una distancia considerable entre la fuente y los reguladores, proporcionando mayor estabilidad en la entrada. Los capacitores a las salidas (C_o) aunque no tienen efecto en la estabilidad, ayudan a la respuesta transitoria del circuito. Por último los diodos son recomendados como respuesta a posibles problemas de enclavamiento (latch-up). Los valores de los capacitores siguen las recomendaciones de la hoja de datos resultando en $C_i=10\mu F$ y $C_o=0,1\mu F$.

El uso de estos reguladores además de proporcionar protección al sistema, permiten que a futuro al hacer el dispositivo totalmente portable con un conjunto de baterías, con simplemente conectarlas como entradas a estos reguladores el sistema funcione de forma correcta. Claramente esto no toma en cuenta el sistema de recarga que sería necesario para dichas baterías sin embargo esto no formaba parte del prototipo desarrollado.

3.6 Sistema de acondicionamiento entero

El orden en el que se realizan las etapas descritas hasta este punto está plasmado en la figura 3.24.

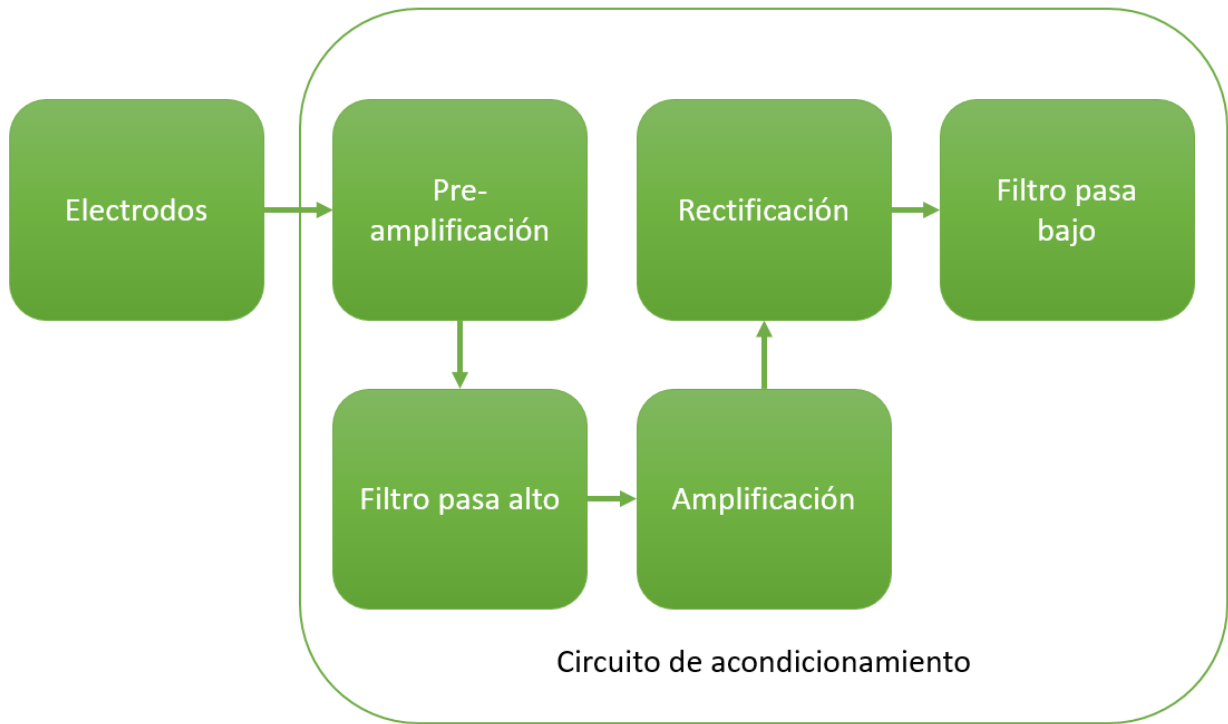


Figura 3.24 Diagrama del sistema de acondicionamiento.

La señal es medida por el arreglo de electrodos, cada par de electrodos va a un circuito de acondicionamiento exactamente igual. Primero se realiza la pre-amplificación, pasa por el filtro pasa alto, es amplificada por segunda vez, se rectifica y finalmente pasa al filtro pasa bajo. El diagrama del circuito entero se puede observar en las figuras 3.25 y 3.26, el cual incluye el sistema de alimentación en un extremo además de que se puede observar la inclusión de un seguidor (buffer) al final del circuito es decir luego del filtro pasa bajo.

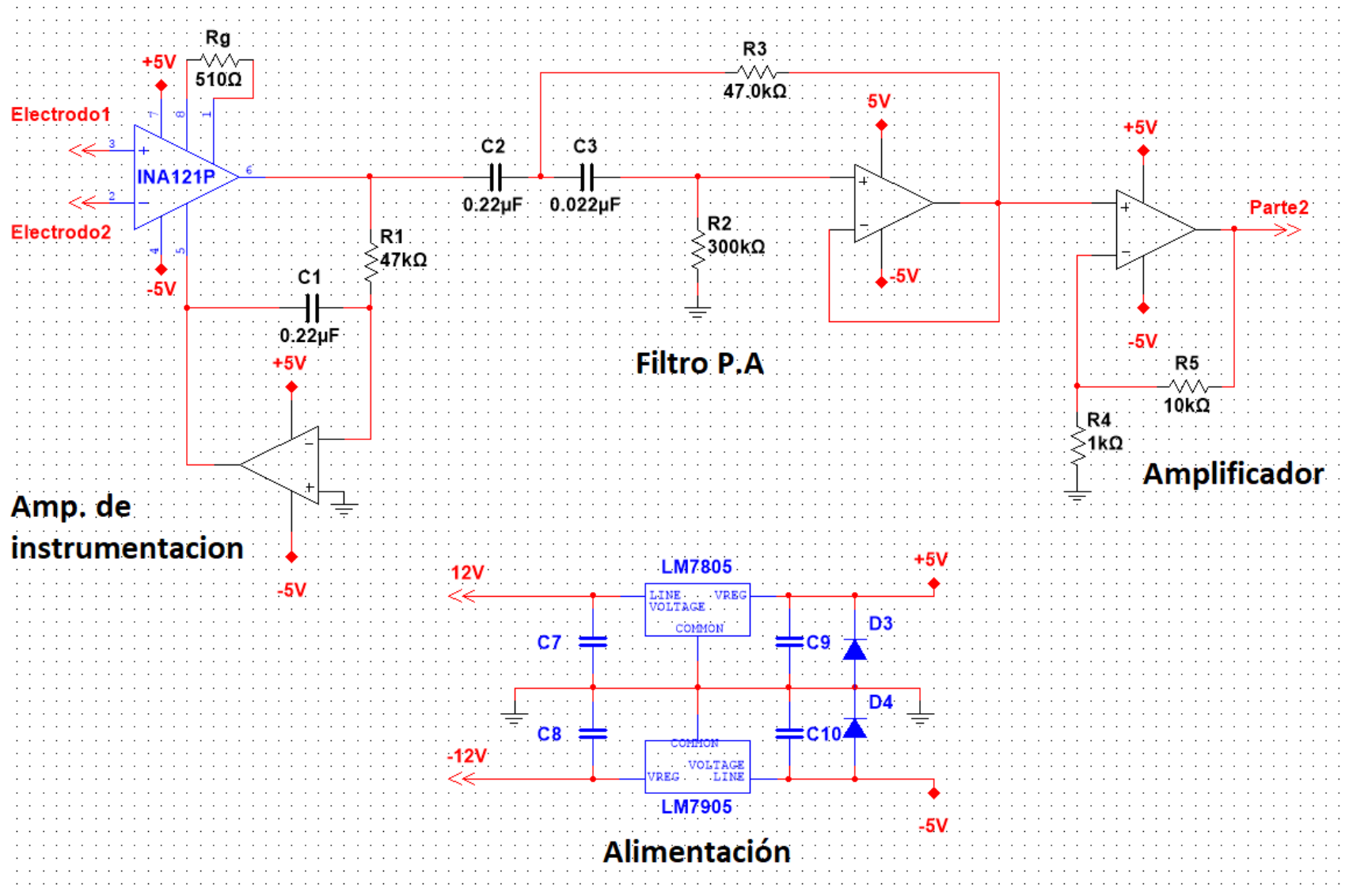


Figura 3.25 Diagrama del circuito de acondicionamiento – 1^{ra} parte.

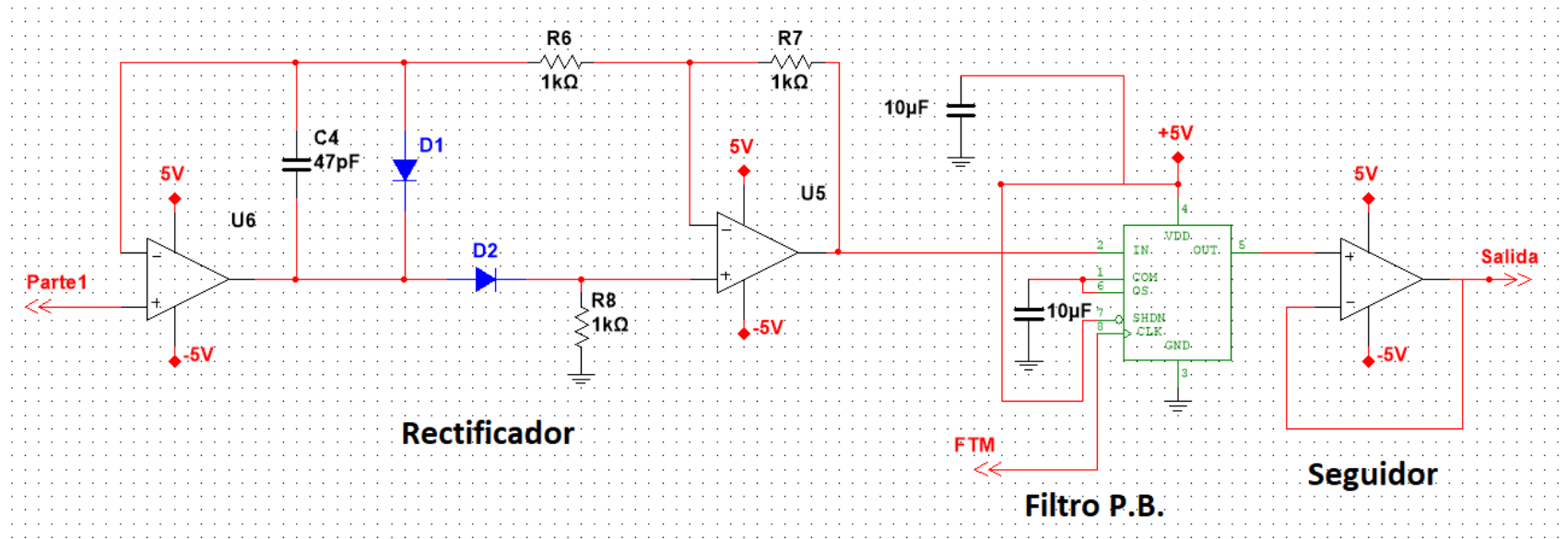


Figura 3.26 Diagrama del circuito de acondicionamiento – 2^{da} parte.

3.7 Resultados de la adquisición

En las siguientes figuras se puede observar la señal medida con un osciloscopio a la salida de cada una de las etapas del circuito de acondicionamiento. En el momento de tomar las imágenes se realizó contracciones de forma que se pudiese observar una señal en sus valores más altos.

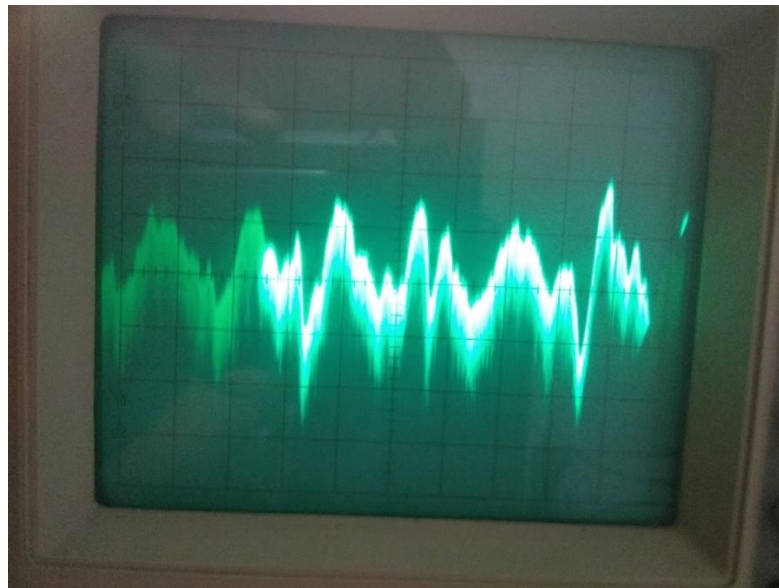


Figura 3.27 Salida del Amplificador de instrumentación – 10mV/div- 10ms/div.

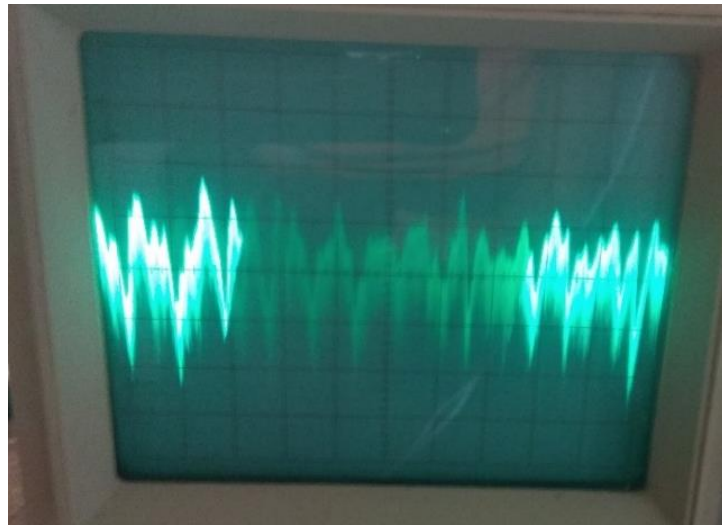


Figura 3.28 Salida del filtro pasa bajo – 10mV/div- 10ms/div.

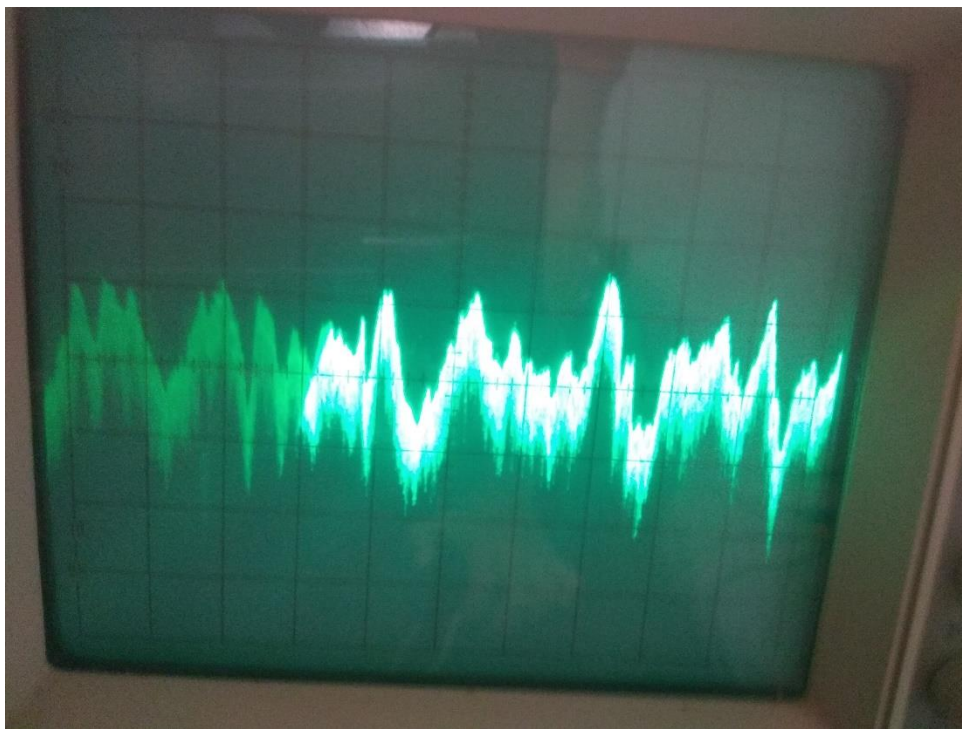


Figura 3.29 Salida del amplificador – 1V/div- 10ms/div.

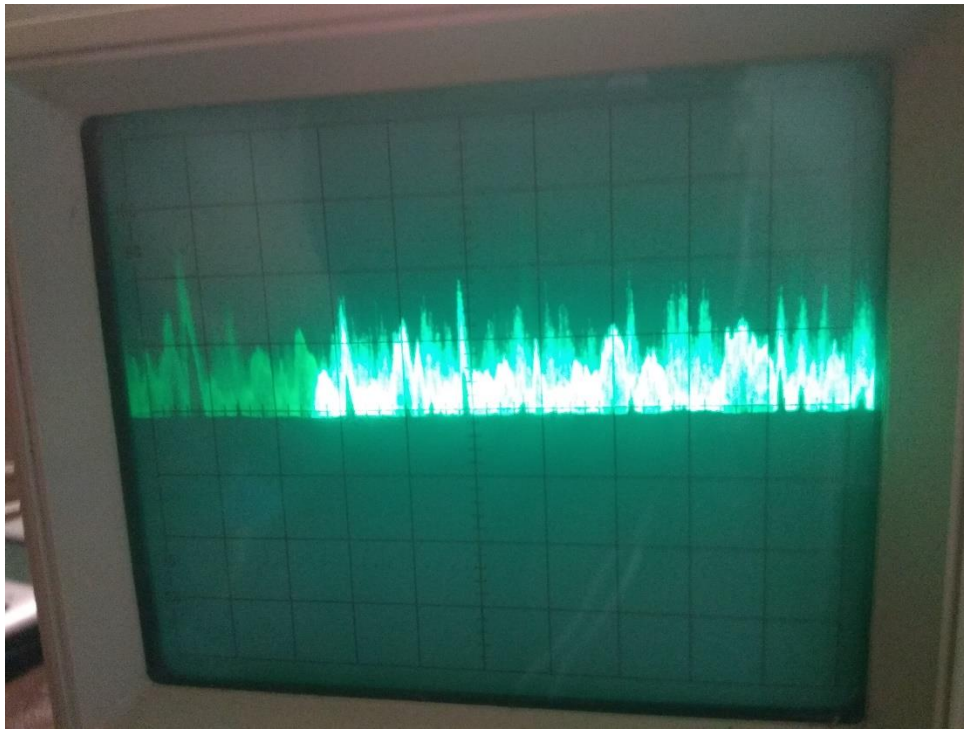


Figura 3.30 Salida del rectificador – 1V/div- 10ms/div.

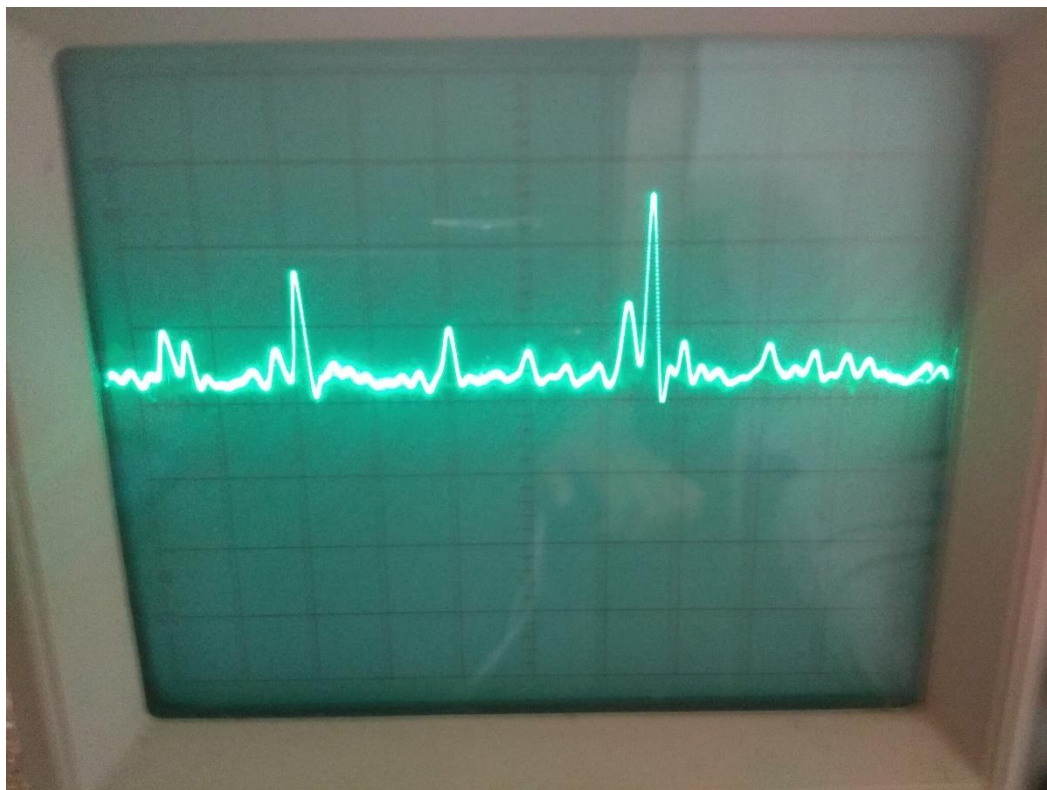


Figura 3.31 Salida del filtro pasa bajo – 1V/div- 10ms/div.

Como se observa en la figura 3.31, a la salida del circuito de acondicionamiento se tiene una señal limpia, unipolar y menor a 3,3V.

En el recorrido se puede observar una fuerte presencia de ruido, el cual es atribuido a dos factores principales; primero todo el circuito está montado en una placa de pruebas (protoboard), mediciones realizadas incluso entre puntos adyacentes presentaban perturbaciones que son esperadas cuando se tiene este tipo de montajes; el segundo factor es la fuente que provee energía al circuito, esta es una fuente constante de vibraciones y oscilaciones que de forma ideal no se tendrían presentes. Sin embargo una futura implementación en la que todo el circuito pase a una tarjeta y en la que el circuito pase a ser alimentado por baterías debería mejorar el resultado, el cual como se observa en la figura 3.27 ya es bueno. De igual forma lo más relevante son los resultados de las etapas posteriores, descritas en los capítulos siguientes, cuyos resultados validan que esta señal fue suficiente dadas las necesidades.

CAPITULO 4

PROCESAMIENTO DE LA SEÑAL

A la salida del circuito de acondicionamiento se tiene una señal EMG entre 0 y 3,3V que es compatible con el microcontrolador. En este punto lo que se necesita es llevar la señal analógica al mundo digital, guardar los datos y analizar la información para extraer las características que alimentan a la red neuronal.

4.1 Adquisición y guardado de datos

Para dejar la información en un estado apto para su manejo (extracción de características) se lleva la señal del dominio analógico al digital. Dado que es un flujo continuo de datos, se trabaja con una ventana de tiempo de un tamaño definido, es decir se realiza la conversión de un número específico de puntos que son guardados en la memoria del procesador, para luego trabajar con dicha señal discreta en la extracción de características.

Realizar este proceso responde a unos parámetros que tienen que ver con la velocidad de trabajo y la forma óptima de manejar la señal. En primera instancia, era necesario que la velocidad del proceso de conversión fuera lo suficientemente rápida para poder discretizar la señal de forma correcta; segundo, la ventana de tiempo debía tener suficientes datos para ser una buena representación de la señal pero teniendo en mente que una cantidad desproporcionada de datos no solo ralentizaría todas las etapas posteriores sino que lo haría sin obtener beneficios.

Ya que en el acondicionamiento de la señal fue establecido que los componentes de más baja frecuencia a utilizar son de 20Hz, esto delimita la ventana de tiempo más pequeña que se podía establecer para poder tener “señal entera”. Esto quiere decir que la ventana de tiempo debía ser de al menos 50ms como se observa en la figura 4.1. Tamaños mucho mayores a este no proporcionarían información relevante tomando en cuenta que este proceso es continuo y que al procesar una ventana de datos inmediatamente se empieza a guardar la siguiente.

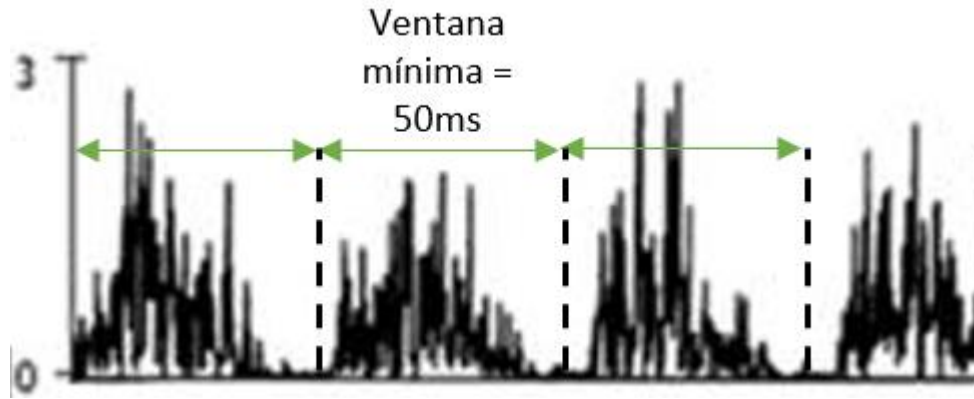


Figura 4.1 Ventana de tiempo mínima.

El microprocesador K22F cuenta con un módulo ADC, con 24 canales de conversión simple. Este puede trabajar con diferentes modos y configuraciones; los parámetros de trabajo utilizados fueron:

- Conversión continua; apenas termina una conversión se procede a la siguiente de forma automática.
- Los datos transformados tienen una resolución de 16bits, la resolución máxima permitida. Esto significa que los 3,3V máximos de entrada son representados en 16bits de forma digital teniendo $2^{16}=65536$ valores posibles.
- El sistema de promedio por hardware encendido, configurado a 32 mediciones (máximo admitido). Este periférico permite tomar un número de mediciones a alta velocidad y dar como resultado de la conversión el promedio de dichas medidas.
- Fuente del reloj proviene del reloj del BUS con un divisor de 2. El reloj del BUS está establecido a una frecuencia de 20,972MHz por lo que el reloj del ADC es de 10.486MHz.
- Se trabaja con 2 canales del ADC, uno para cada señal EMG.

A partir de estos parámetros se puede estimar el tiempo de conversión teórico siguiendo lo indicado en la manual de referencia del microcontrolador [23].

$$Reloj\ del\ ADC\ (ADCK) = \frac{Reloj\ del\ bus}{Divisor} = \frac{20,972MHz}{2} = 10,486MHz \quad (4.1)$$

$$Ciclo.BUSCLK = \frac{1}{20,972MHz} \approx 0,0477\mu s \quad (4.2)$$

$$Ciclo.ADCK = \frac{1}{10,486MHz} \approx 0,0957\mu s \quad (4.3)$$

$$T.Conv = Sum.UC + N.Prom * (TCB + Sum.ML + Sum.MAV) \quad (4.4)$$

Sum.UC: Sumador de tiempo por conversión única o primera conversión continua. Este elemento toma en cuenta el tiempo necesario para iniciar conversión, depende si está configurada conversión larga y cuál es el reloj fuente. Para este caso:

$$\begin{aligned} Sum.UC &= 5 * Ciclos.ADCK + 5 * Ciclos.BUSCLK \\ &=> 5 * 0,0957\mu s + 5 * 0,0477\mu s \approx 0,7152\mu s \end{aligned} \quad (4.5)$$

N.Promedio: factor de número promedio. Si el promedio por hardware esta encendido multiplica el tiempo de medición por cantidad de medidas a tomar en cuenta.

$$N.Promedio = 32 \quad (4.6)$$

TCB: tiempo de conversión base. Cuantos ciclos de reloj toma la conversión dependiendo de la resolución de los resultados.

$$TCB = 25 * Ciclos.ADCK \approx 2\mu s \quad (4.7)$$

Sum.ML: sumador de tiempo por medida larga. Si se están realizando medidas largas se agregan unos ciclos de reloj extra por cada conversión realizada.

Sum.MAV: sumador de tiempo por conversión de alta velocidad. Si se están realizando medidas a alta velocidad se agregan par de ciclos del reloj.

$$Sum.ML \& Sum.MAV = 0 \quad (4.8)$$

Finalmente, el tiempo de conversión teórico es de:

$$T. Conversion = 0,7152\mu s + 32 * (2\mu s + 0) \approx 77\mu s \quad (4.9)$$

Esto resulta en una frecuencia de conversión de 12,987KHz.

Dado que se necesitaba una ventana de al menos 50ms, se decidió guardar 1000 conversiones del ADC lo cual representa una ventana de aproximadamente 77ms.

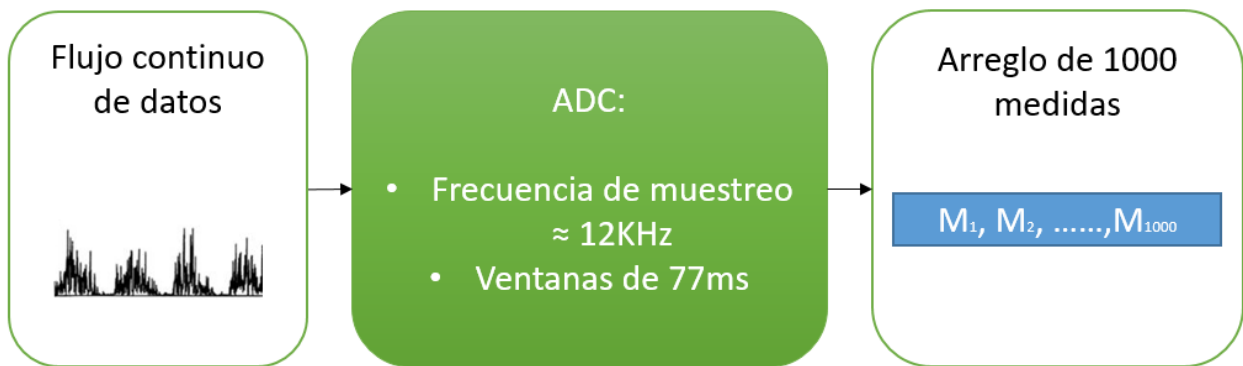


Figura 4.2 Diagrama de adquisición de datos.

4.2 Extracción de características

En este punto se tiene una representación de la señal en forma de 1000 medidas guardadas en un arreglo del mismo tamaño. Para poder alimentar la red neuronal fue necesario extraer una serie de características, obteniendo información a partir de las propiedades de la señal EMG. Entre más información relevante existe en las características elegidas, la red será más eficiente tanto porque será capaz de reconocer mejor los patrones como porque necesitara menos características de entrada lo cual agiliza todo el proceso.

Dado que es un flujo constante de datos, fue necesario elegir un microcontrolador que tuviese sistema procesamiento digital de señales o DSP por sus siglas en inglés, [27] para poder manejar el alto volumen de datos con la velocidad necesaria para hacer el proceso en tiempo real. Este tipo de procesadores posee un conjunto de instrucciones tanto a nivel de software como hardware diseñado específicamente para aplicaciones que necesiten realizar operaciones matemáticas a alta velocidad, y esto es exactamente lo que se requería en este proyecto.

La extracción de características conlleva la ejecución de un conjunto de algoritmos matemáticos sobre los datos guardados y además se necesita que luego toda esta información pase por la red neuronal lo cual implica aún más cálculos. Cuando se toma todo esto en cuenta la velocidad de procesamiento de datos es lo primordial. Por esta razón se utilizó el microprocesador K22F el cual cuenta con un núcleo ARM Cortex - M4 con instrucciones DSP.

En el análisis de señales EMG se pueden hacer estudios en el tiempo, estudios frecuenciales o combinaciones de estos. En este proyecto cada uno de los algoritmos elegidos para la obtención de características, son relaciones matemáticas que estudian la señal en el tiempo.

Aunque algunos de los algoritmos son sencillos y solo requieren una operación matemática (desde el punto de vista de instrucción DSP), otros requieren un conjunto de operaciones. Este fue uno de los parámetros en la elección de algoritmos; se necesitaba que dieran la información suficiente para que la red neuronal descubriera patrones, pero que su implementación fuese relativamente fácil. De nada servía recrear un algoritmo que requiera decenas de líneas de código y mucho más tiempo de ejecución si con uno sencillo era suficiente.

Las características elegidas son: media de la señal, la varianza, longitud de forma de onda, amplitud de Willison y cambios de signo de la pendiente.

4.2.1 Media de la señal

Esto es simplemente el valor promedio de la señal, se toman todos los datos del arreglo y se dividen entre el número de elementos.

$$Media = \frac{1}{N} \sum_{i=1}^N |X_i| = \frac{1}{1000} \sum_{i=1}^{1000} X_i \quad (4.10)$$

Implementar este cálculo en el programa es tan sencillo como llamar a la función de promedio existente en las librerías DSP [27].

4.2.2 Varianza

La varianza es una medida estadística de dispersión para una variable aleatoria. Una forma de describirla es que tan lejos están distribuidos un grupo de elementos a partir de su valor medio. Estudios indican que la varianza en señales EMG son una representación de la densidad de potencia en el músculo [28].

$$Var = \sigma^2 = \frac{1}{N-1} \sum_{i=1}^N X_i^2 \quad (4.11)$$

El cálculo de la varianza esta implementado en las librerías DSP como una función única [27].

4.2.3 Cambios de signo de la pendiente

Cuenta el número de veces que la pendiente de la señal cambia de signo. Requiere un umbral para evitar que el ruido sea factor.

$$(X_i - X_{i-1}) * (X_i - X_{i+1}) \geq umbral ; Con i = 1, \dots, 1000 \quad (4.12)$$

Para realizar este cálculo se necesita además del arreglo de las medidas, dos arreglos auxiliares con las medidas desplazadas. Esto significa que aparte de tener el arreglo de 1000 elementos con todas las medidas, se crean dos arreglos paralelos en los cuales todos los datos son iguales al original pero están desplazados un elemento a la izquierda y a la derecha respectivamente como se puede observar en la figura 4.3. En cada caso un arreglo tiene el primer elemento siempre 0 y el otro arreglo siempre el último elemento siempre 0. El armado de estos dos arreglos se realiza al tomar las medidas, es decir cuando el ADC realiza una medición lo guarda en su posición respectiva en cada uno de los tres arreglos.

Medidas	M ₁	M ₂	M ₃	...	M ₁₀₀₀
M. Desplazadas a la derecha	0	M ₁	M ₂	...	M ₉₉₉
M. Desplazadas a la izquierda	M ₂	M ₃	M ₄	...	0

Figura 4.3 Ejemplo de arreglos auxiliares.

El cálculo de esta característica consiste en un simple ciclo que recorre los arreglos comprobando que la resta y multiplicación de los respectivos elementos, de mayor que el umbral especificado.

4.2.4 Longitud de forma de onda (Waveform Length)

Esta característica provee una medida de la complejidad de la señal. Se define como la longitud cumulativa de la señal en la ventana de estudio [29]. Una forma sencilla de verlo es calcular la distancia entre cada uno de los puntos adyacentes en la señal discreta, calculando todo el recorrido en la superficie de la señal.

$$WL = \sum_{i=1}^{1000} |\Delta X_i|; \text{ donde } \Delta X_i = X_i - X_{i-1} \quad (4.13)$$

En este caso el algoritmo se construye en 3 pasos, haciendo uso del arreglo original y el arreglo desplazado a la derecha descrito en la sección anterior:

1. Se restan estos dos arreglos elemento por elemento y el resultado es un arreglo de salida.
2. Se calcula el valor absoluto.
3. Se realiza la suma de todos estos elementos a partir de la función producto punto con un arreglo auxiliar de puros 1.

4.2.5 Amplitud de Willison

Ésta cuenta el número de veces que la amplitud de la señal cambia y que es mayor que un umbral específico. En las señales EMG este elemento está relacionado con el potencial en la activación de las unidades motoras y permite estimar el nivel de contracción de un músculo [29].

$$WAMP = \sum_{i=1}^{1000} f(|X_i - X_{i+1}|); f(x) = \begin{cases} 1, & \text{si } x > \text{umbral} \\ 0, & \text{cualquier otro caso} \end{cases} \quad (4.14)$$

De forma similar a la longitud de onda, para este cálculo se hace necesario uno de los arreglos desplazados, en este caso el arreglo desplazado a la izquierda.

1. Se restan estos dos arreglos, elemento por elemento y el resultado es un arreglo de salida.
2. Se calcula el valor absoluto.
3. Se compara cada elemento con el valor del umbral, si este es mayor una variable es aumentada en 1. El resultado del ciclo entero es una variable con el número de veces que la resta fue mayor al umbral.

Todos estos algoritmos dan como resultado valores numéricos flotantes, lo que significa que al final de la extracción de características se tienen 5 valores numéricos que pasaran de entrada a la red neuronal.

El valor de los umbrales que se utilizan para la amplitud de Willison y para los cambios de signo de la pendiente, están basados en inicio por estudios como los descritos en [29] pero fueron variados de forma empírica. Observando en tiempo real la variación de los valores de cada característica con los gestos realizados se aumentó o disminuyó los valores iniciales hasta que se tenían resultados considerados óptimos. Al final de las pruebas los valores para los umbrales quedaron:

- Cambios de signo de la pendiente: 0,00001. Este no posee unidades ya que es una medida de la pendiente.
- Amplitud de Willison: 10mV.

CAPITULO 5

RED NEURONAL

Con el conjunto de características extraídas de las señales EMG, lo que se necesita es reconocer patrones en la información para poder separar diferentes comportamientos (gestos en este caso). Esta búsqueda de patrones en los datos por métodos convencionales (observación) sería extremadamente difícil e ineficiente, por lo que enseñar a una computadora a encontrar los patrones que los humanos no vemos fácilmente es una excelente solución.

Emplear aprendizaje supervisado es posible ya que se conocen los gestos (salida) a los que corresponden las muestras (entrada). El trabajo del algoritmo de aprendizaje es, a partir de estos ejemplos, armar el “programa” que discernirá las salidas que corresponden a cada entrada.

Lo que se implementó en este proyecto es una red neuronal que puede ser programada cada vez que se quieren utilizar distintos o más gestos, por diferentes personas. La red neuronal implementada fue diseñada con la ayuda de la caja de herramientas de redes neuronales de MATLAB. En esta se tienen diferentes tipos de redes neuronales entre las cuales se encuentra una de búsqueda de patrones. Esta es una red prealimentada, entrenada por un algoritmo de propagación hacia atrás [30].

La idea es que una vez entrenada la red y que esta ha sido recreada en el microprocesador, a partir del flujo constante de características que son extraídas con cada medición del ADC (conversor analógico digital) la red determina que gesto se está realizando.

5.1 Entrenamiento

Para poder realizar el entrenamiento de la red neuronal se necesitan tener el conjunto de datos en un formato específico. Los datos son las características extraídas de las señales obtenida por el ADC. Dichos datos deben estar ordenados en una matriz de $C \times M$ elementos donde C es el número de características y M el número de muestras a pasar como entrenamiento. Además, al ser una red de entrenamiento supervisado necesita una matriz de $G \times M$ elementos donde G es el

número de posibles resultados, en este caso cuantos gestos están siendo programados. Un ejemplo de estas matrices de entrenamiento se tiene en la figura 5.1.

A_{11}	A_{12}	...	A_{1M}
A_{21}	A_{22}	...	A_{2M}
...
A_{C1}	A_{C2}	...	A_{CM}

B_{11}	B_{12}	...	B_{1M}
B_{21}	B_{22}	...	B_{2M}
...
B_{G1}	B_{G2}	...	B_{GM}

Figura 5.1 Matrices de entrenamiento.

Cada elemento A_{ij} de la matriz de muestras es una característica de una muestra específica es decir un número. Por otro lado los elementos B_{ij} son representaciones probabilísticas de que la muestra correspondiente sea ese gesto. Como en el entrenamiento se conocen los gestos realizados en cada muestra, cada columna tiene un elemento que vale 1 (100% de probabilidad que es el gesto realizado) mientras que el resto vale 0.

Para explicarlo con un ejemplo digamos que se está entrenando una red en la cual se tienen 3 posibles resultados, es decir 3 gestos. La salida de la red por cada entrada son 3 números, donde cada número corresponde a la probabilidad de que se esté ejecutando el gesto G . Estos números son probabilidades normalizadas por lo que solo pueden variar de 0 a 1 y la suma de ellos es siempre 1. En la figura 5.2 se tiene un ejemplo de la matriz de resultados que se debería pasar para este ejemplo.

B_{11}	B_{12}	...	B_{1M}		0	1	...	1
B_{21}	B_{22}	...	B_{2M}		0	0	...	0
B_{31}	B_{32}	...	B_{3M}		1	0	...	0

Figura 5.2 Ejemplo de matriz de resultados para entrenamiento.

Al tener 3 gestos, cada muestra solo puede corresponder a uno de ellos. En el ejemplo de la figura 5.2 la primera muestra corresponde al tercer gesto así que el resultado es el vector columna 0/0/1; de forma similar la segunda muestra y la última corresponden al primer gesto es decir su respuesta debe ser 1/0/0.

Cuando la red da un resultado, estos tres números responden a la probabilidad que estima la red que los datos de entrada correspondan a dicha casilla. Por ejemplo si la red da como salida 0,1/0,8/0,1, esto significa que la red estima que hay un 80% de posibilidades que el gesto realizado sea el segundo.

Para el armado de las dos matrices de entrenamiento, la de entrada o muestras y la de salida, no hay un procedimiento directo, se necesita una serie de pasos.

Primero extraen las características de las señales del microprocesador. Esto significa enviar las características por comunicación serial. En la figura 5.3 se puede observar una representación gráfica de 5 características siendo recibidas; dicha grafica se puede obtener gracias al programa SerialPlot [31].

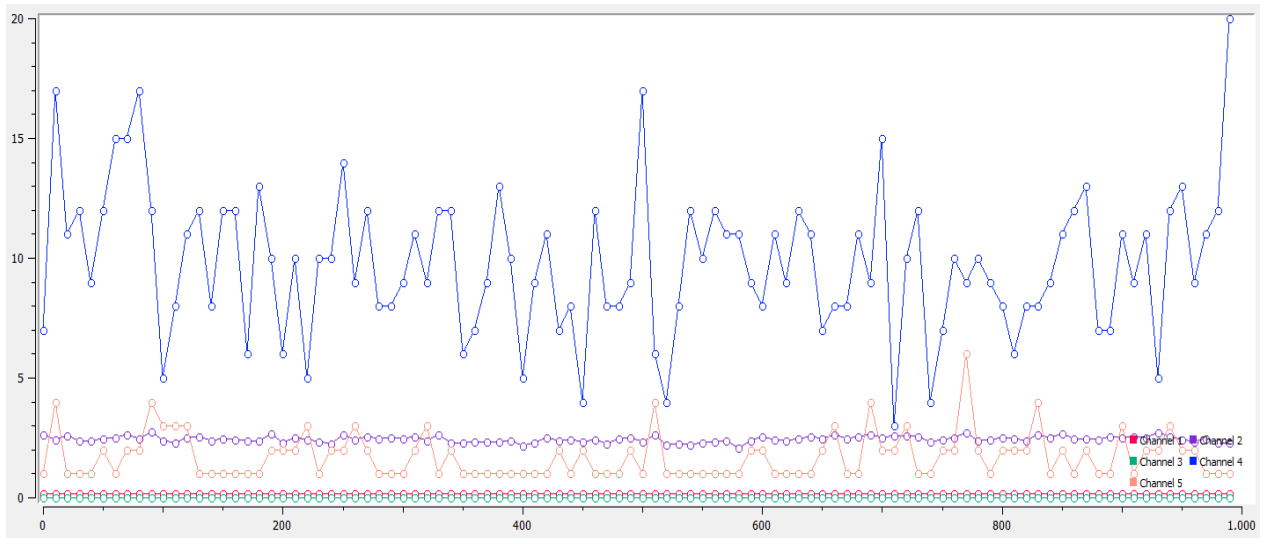


Figura 5.3 5 características siendo recibidas por serial.

Luego se deben guardar las muestras. Los datos recibidos deben ser guardados para poder armar con ellas la matriz de entrada. SerialPlot además de mostrar gráficamente valores recibidos por serial permite guardar dicha información. En este caso los datos fueron guardados en archivos de valores separados por coma o “.CSV” por sus siglas en ingles. Esto significa que cada muestra que consta de 10 características (5 por cada señal EMG medida) queda representada de la siguiente forma $[C_1, C_2, C_3, \dots, C_{10}]$. Un archivo “.CSV” abierto en Microsoft Excel se ve como en la figura 5.4, donde cada fila representa una medición.

	A	B
1	0.176563, 2.36714, 0.000185, 3, 3, 0.280821, 3.74135, 0.000615, 2, 24	
2	0.17087, 2.17293, 0.000166, 5, 1, 0.269834, 3.03116, 0.000475, 0, 6	
3	0.167793, 2.22303, 0.000123, 1, 1, 0.266568, 3.0939, 0.000381, 3, 15	
4	0.1657, 2.12252, 0.000139, 6, 1, 0.264184, 3.01485, 0.000415, 5, 2	
5	0.168414, 1.9702, 0.00011, 3, 1, 0.268532, 3.21304, 0.000391, 4, 3	
6	0.169792, 2.1903, 0.00015, 2, 1, 0.270324, 3.18368, 0.000447, 1, 10	
7	0.169808, 2.05535, 0.000129, 3, 1, 0.26838, 3.26682, 0.000419, 1, 4	
8	0.168799, 1.93843, 0.000136, 2, 1, 0.267904, 3.1795, 0.000438, 2, 5	
9	0.166399, 1.87936, 0.000121, 5, 1, 0.264215, 2.866, 0.000396, 0, 3	
10	0.164749, 2.08818, 0.000139, 6, 1, 0.260728, 3.12099, 0.000464, 1, 4	
11	0.167263, 2.13184, 0.000157, 0, 1, 0.264716, 2.91041, 0.000379, 3, 2	
12	0.166822, 2.25949, 0.000148, 4, 1, 0.261736, 3.03761, 0.000379, 1, 11	
13	0.164406, 2.03798, 0.000123, 9, 1, 0.258765, 3.03493, 0.000459, 1, 11	
14	0.1677, 2.09931, 0.000173, 0, 1, 0.26346, 3.19109, 0.000441, 2, 4	
15	0.167182, 2.10611, 0.000152, 5, 1, 0.262636, 3.09793, 0.000467, 4, 8	
16	0.165437, 2.15767, 0.00015, 4, 1, 0.260786, 3.18595, 0.000428, 6, 6	
17	0.165681, 1.92181, 0.000152, 3, 1, 0.26111, 2.90537, 0.000472, 2, 1	
18	0.165999, 1.89628, 0.000109, 2, 1, 0.263124, 3.15876, 0.000383, 0, 2	
19	0.165772, 2.14211, 0.000145, 4, 1, 0.261823, 3.18514, 0.000419, 1, 8	
20	0.164272, 1.97423, 0.000115, 5, 1, 0.260642, 2.70114, 0.000343, 3, 2	

Figura 5.4 Ejemplo de características guardadas en archivo “.CSV”.

MATLAB es capaz de abrir los archivos “.CSV” e interpretarlos como una matriz, por esta razón el único requerimiento es que los datos deben ser transpuestos para quedar como una matriz 10xM como en la figura 5.5, compatible con el algoritmo de entrenamiento.

10x3200 double

	1	2	3	4	5	6
1	0.1766	0.1709	0.1678	0.1657	0.1684	0.
2	2.3671	2.1729	2.2230	2.1225	1.9702	2.
3	1.8500e-04	1.6600e-04	1.2300e-04	1.3900e-04	1.1000e-04	1.5000
4	3	5	1	6	3	
5	3	1	1	1	1	
6	0.2808	0.2698	0.2666	0.2642	0.2685	0.
7	3.7414	3.0312	3.0939	3.0149	3.2130	3.
8	6.1500e-04	4.7500e-04	3.8100e-04	4.1500e-04	3.9100e-04	4.4700
9	2	0	3	5	4	
10	24	6	15	2	3	

Figura 5.5 Ejemplo de matriz de entrada para entrenamiento.

La matriz de salida si es armada desde cero. Por esta razón puede ser elaborada por diferentes métodos como por ejemplo crear un archivo “.CSV” similar a las entradas donde cada fila es la salida esperada como se observa en la figura 5.6. Otra opción es directamente armar la matriz en MATLAB, resultando en una matriz como la figura 5.7.

	A	B
1	1,0,0,0	
2	1,0,0,0	
3	1,0,0,0	
4	1,0,0,0	
5	1,0,0,0	
6	1,0,0,0	
...		
3182	0,0,0,1	
3183	0,0,0,1	
3184	0,0,0,1	
3185	0,0,0,1	
3186	0,0,0,1	
3187	0,0,0,1	
3188	0,0,0,1	
3189	0,0,0,1	

Figura 5.6 Ejemplo de datos de salida para entrenamiento.

4x3200 double							
	1	2	3				
1	1	1	1	• • •	3198	3199	3200
2	0	0	0		0	0	0
3	0	0	0		0	0	0
4	0	0	0		1	1	1

Figura 5.7 Ejemplo de matriz de salidas para entrenamiento

Las matrices de entrada y salida, y el número de neuronas por el cual estará compuesta la red, son los elementos mínimos necesarios para que MATLAB pueda realizar un entrenamiento. El número de muestras y la calidad de estas determinan si la red será capaz de trabajar correctamente una vez entrenada, entre más muestras mejor. En las pruebas realizadas se tomaron

las medidas de 4 gestos durante poco menos de 2 minutos cada uno, que luego se ordenaron en 800 medidas de cada gesto, dando un total de 3200 elementos para el entrenamiento. En cuanto al número de neuronas no hay forma exacta de estimar el valor que sea más acorde para cada caso, por lo que fue necesario realizar el entrenamiento con varios valores y comparar el desempeño de cada uno. En esta implementación se utilizaron en 10 neuronas en la capa oculta.

Solo se utilizó una capa oculta ya que capas extra, que corresponderían a una red neuronal profunda, son utilizada en problemas mucho más complejos además significaría escalar fuertemente el trabajo que se realiza en el microprocesador.

Los resultados del entrenamiento o la red entrenada, consiste en todos los pesos de las conexiones de las neuronas. Cada uno de estos elementos se debe llevar al microprocesador para ser ordenados y puestos en un formato que permita ser utilizado.

Aunque el entrenamiento en sí, solo toma unos segundos, todo el procedimiento de capturar un conjunto de medidas y llevarlas al formato que requiere el entrenamiento (crear las matrices), es un proceso relativamente largo y no amigable. Es por esto que la continuación de este proyecto a futuro tendría como prioridad desarrollar un programa que permita un entrenamiento automatizado.

El diagrama de la red neuronal diseñada se puede observar en la figura 5.8.

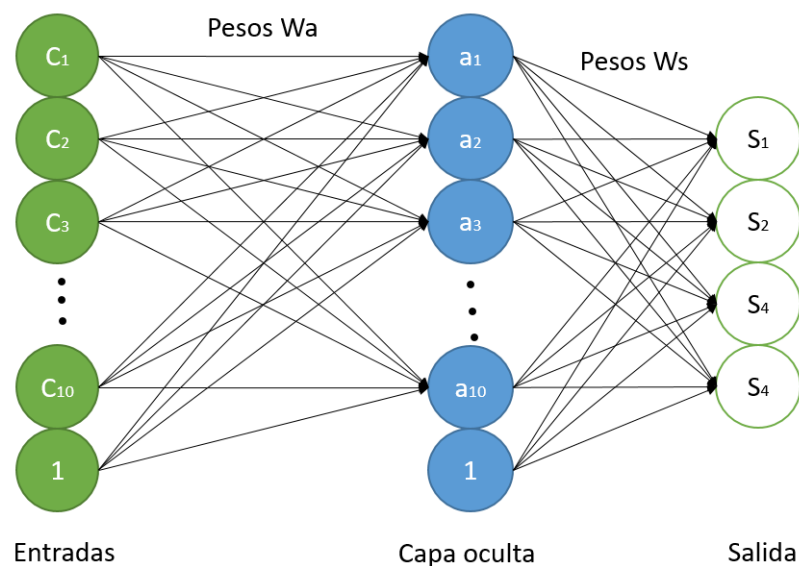


Figura 5.8 Diagrama de la red neuronal diseñada.

Un diagrama simplificado del proceso de entrenamiento se puede observar en la figura 5.9.

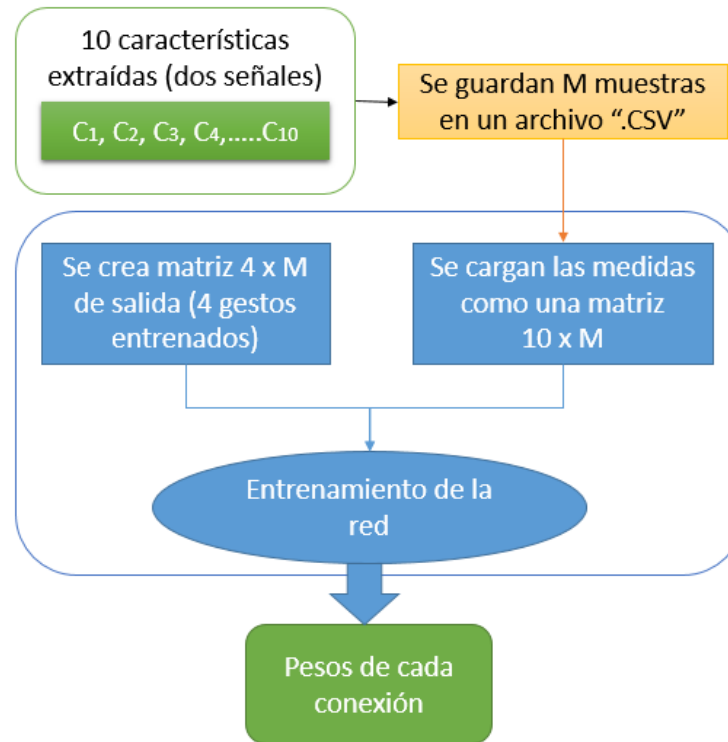


Figura 5.9 Diagrama del proceso de entrenamiento.

5.2 Implementación

Dentro del microcontrolador la ejecución de la red neuronal consiste en el paso de las 10 características por un conjunto de operaciones matemáticas que da como resultado G números. Dado el entrenamiento realizado, $G = 4$.

Para manejar el conjunto de datos de forma eficiente, el sistema DSP permite que todos los arreglos numéricos se puedan interpretar como una matriz. Para crear una matriz solo se debe indicar la dirección en memoria del arreglo, el número de filas y columnas. Con este nuevo elemento matriz se pueden realizar operaciones de matrices como multiplicaciones o sumas.

La existencia de matrices permite el armado de la red neuronal de una forma sencilla de entender.

Se tiene un vector de entrada que denominamos V.E, con 11 elementos correspondientes a las 10 características y el bias de la entrada. Luego una matriz 10x11 que contiene los pesos de todas las conexiones de las entradas a la capa oculta denominada W_a , donde las filas son por las 10 neuronas en la capa oculta y 11 columnas que son, 10 por características y una por el bias de la entrada. Y una tercera matriz para la capa de salida 4x11 llamada W_s , que posee 4 filas por que se tienen 4 elementos como salida de la red y 11 columnas por las 10 neuronas más los pesos para el bias de la capa oculta. Representaciones graficas de estas tres matrices se tienen en las figuras 5.10, 5.11, 5.12.

V.E.

C_1
C_2
...
C_{10}
$B=1$

Figura 5.10 Vector de entrada.

W_a

$W_{1,1}$	$W_{1,2}$...	$W_{1,11}$
$W_{2,1}$	$W_{2,2}$...	$W_{2,11}$
...
$W_{10,1}$	$W_{10,2}$...	$W_{10,11}$

Figura 5.11 Matriz con pesos entrada - capa oculta.

W_s

$W_{1,1}$	$W_{1,2}$...	$W_{1,11}$
$W_{2,1}$	$W_{2,2}$...	$W_{2,11}$
$W_{3,1}$	$W_{3,2}$...	$W_{3,11}$
$W_{4,1}$	$W_{10,2}$...	$W_{10,11}$

Figura 5.12 Matriz con los pesos capa oculta - salida.

Los datos fluyen de forma que $W_a * V.E.$ da un vector de 10 elementos los cuales deben pasar por la función de transferencia para dar los valores de las 10 neuronas. La función de transferencia puede ser configurada al momento de realizar el entrenamiento de la red. Entre las disponibles se mantuvo las funciones que dan el mejor resultado y que pudiesen ser recreadas en el microcontrolador. La función de transferencia de la capa oculta es Tansig [32] cuya forma se puede apreciar en la figura 5.13. Esta es una función suave que da como salida un valor real limitado entre -1 y 1.

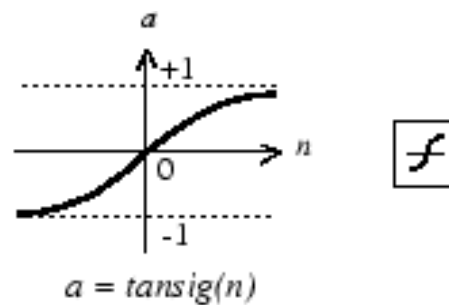


Figura 5.13 Función Tansig [17].

Su forma es similar a una tangente hiperbólica y de hecho sus valores son similares, sin embargo Tansig es mucho más rápida de calcular desde un punto de vista de computo, lo que la hace ideal para una red neuronal donde la rapidez es mucho más importante que la forma exacta de la función.

Dado que esta función no existe de forma nativa en las librerías DSP, fue recreada a partir de su fórmula.

$$tansig(x) = \frac{2}{1 + e^{-2*x}} - 1 \quad (5.1)$$

Cada elemento del vector se pasa por esta función resultando en las 10 neuronas de la capa oculta. El diagrama de esta operación se puede observar en la figura 5.14.

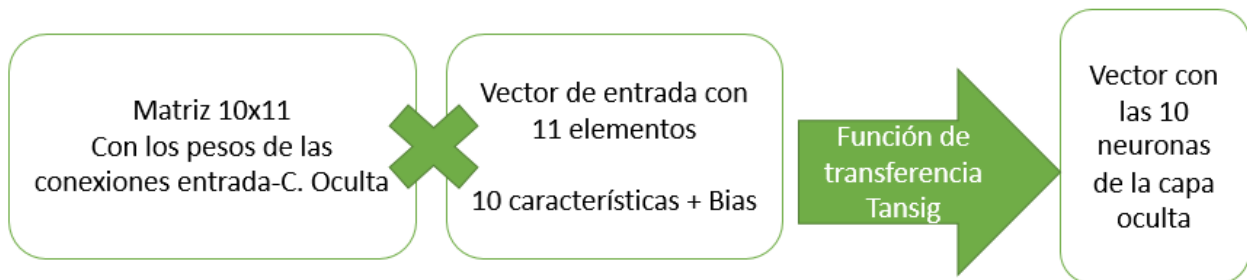


Figura 5.14 Diagrama del cálculo de las 10 neuronas de la capa oculta.

Ya que el vector tiene 10 elementos y se necesita un vector de 11 que incluya el bias de la capa oculta para ser compatible con la siguiente matriz, se crea un vector más grande con su primer elemento igual a 1.

El siguiente paso es multiplicar la matriz W_s por este vector de 11 elementos. Esto resulta en un vector de 4 números que necesita pasar por la función de transferencia de salida que en este caso es Softmax [33]. Un ejemplo de esta se puede ver en la figura 5.15.

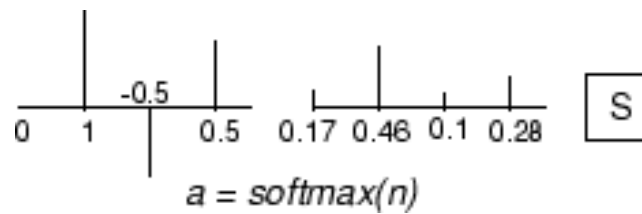


Figura 5.15 Función Softmax [18].

La función Softmax es la función exponencial normalizada. Esta permite comprimir los valores de un vector (valores reales arbitrarios) al convertirlos en valores reales que se encuentran en el rango de 0 a 1. En teoría probabilística esto representa una distribución de probabilidad de posibles salidas. Al ser aplicada como función de transferencia de salida en una red neuronal causa que los resultados de esta sean una representación probabilística de que el resultado corresponda y sea correcto según sus criterios. De nuevo, la función no existe de forma nativa por lo que es recreada a partir de la ecuación 5.2.

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{i=0}^N e^{x_i}} \quad (5.2)$$

Finalmente se tienen 4 números que son el resultado de la red neuronal. El diagrama simplificado de esta operación se puede observar en la figura 5.16. Estos números van de 0 a 1 y pueden ser enviados de forma continua por serial para observar los resultados de la red durante su ejecución.

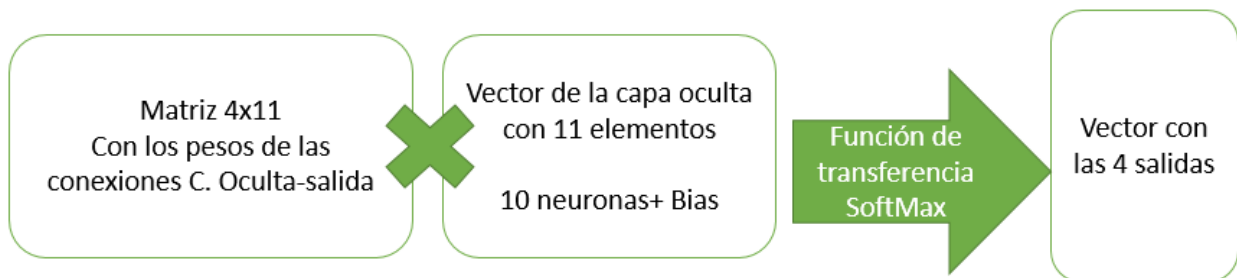


Figura 5.16 Diagrama del cálculo de las 4 salidas.

5.3 Resultados del entrenamiento de la red neuronal

Al realizar el entrenamiento en MATLAB, este proporciona estadísticas del desempeño de la red en cuanto a los datos de entrenamiento proporcionados. MATLAB por defecto calcula el rendimiento de la red a partir de entropía cruzada [8].

En el entrenamiento hecho los 4 gestos eran: mano descansando, mano hacia la derecha, mano hacia la izquierda y mano completamente abierta. Dicho entrenamiento conto con 3200 muestras, 800 para cada gesto. De acuerdo a MATLAB el desempeño esta alrededor de 0.1, no se tiene un valor fijo ya que en cada corrida las muestras que se usan para entrenar y las usadas para probar el desempeño son elegidas al azar. Como se explico en la sección 1.4.1, ya que el desempeño es calculado a partir de entropía cruzada, valores pequeños representan un bajo error, por lo que un resultado cercano a 0 como el obtenido representa un buen desempeño de la red y por tanto del entrenamiento.

Una vez recreada la red en el microprocesador y tener todo el sistema funcionando, se pueden observar los resultados que son enviados por serial para los 4 gestos en las figura 5.17, 5.18, 5.19 y 5.20.

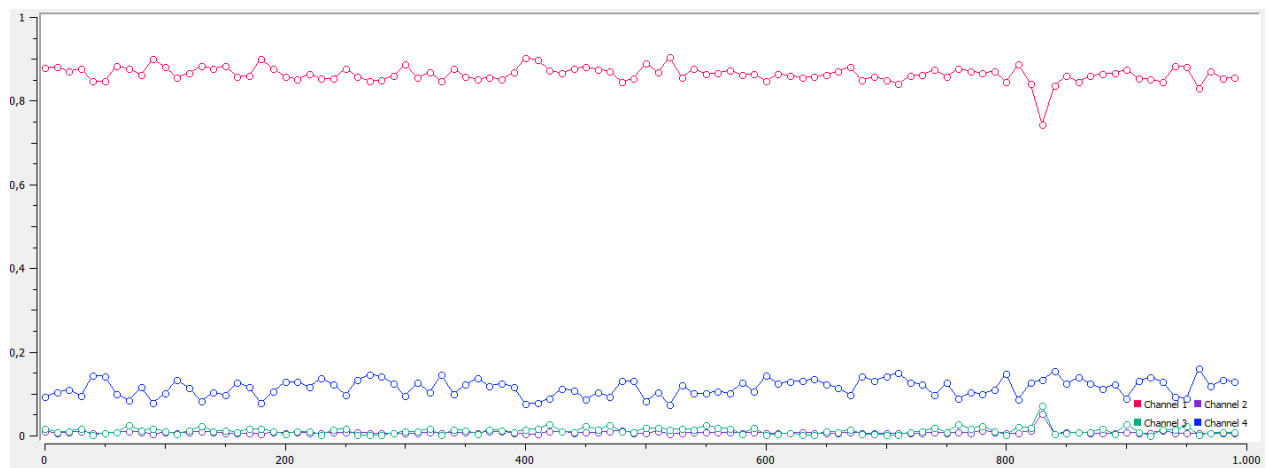


Figura 5.17 Resultado con mano descansando.

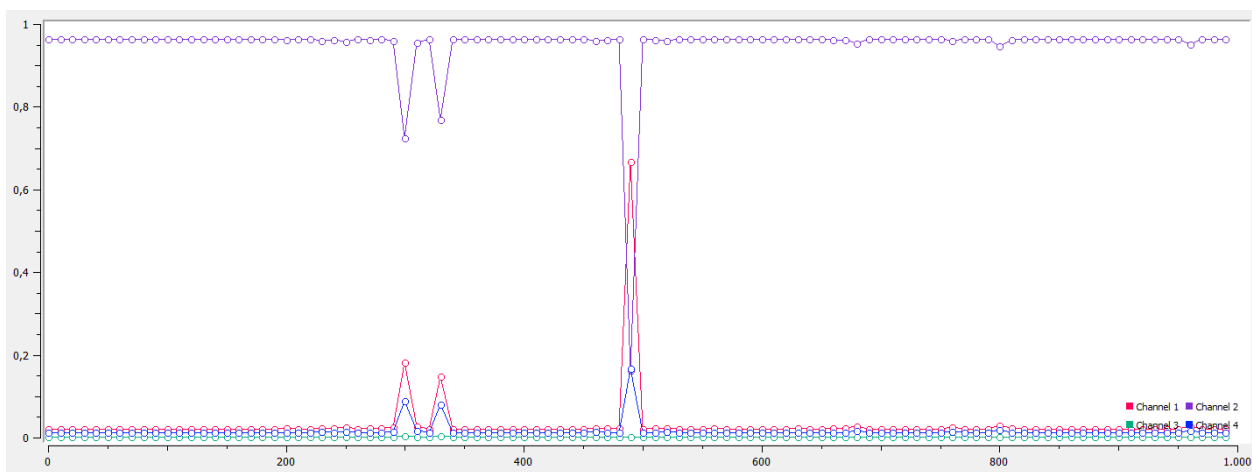


Figura 5.18 Resultado con mano hacia la derecha.

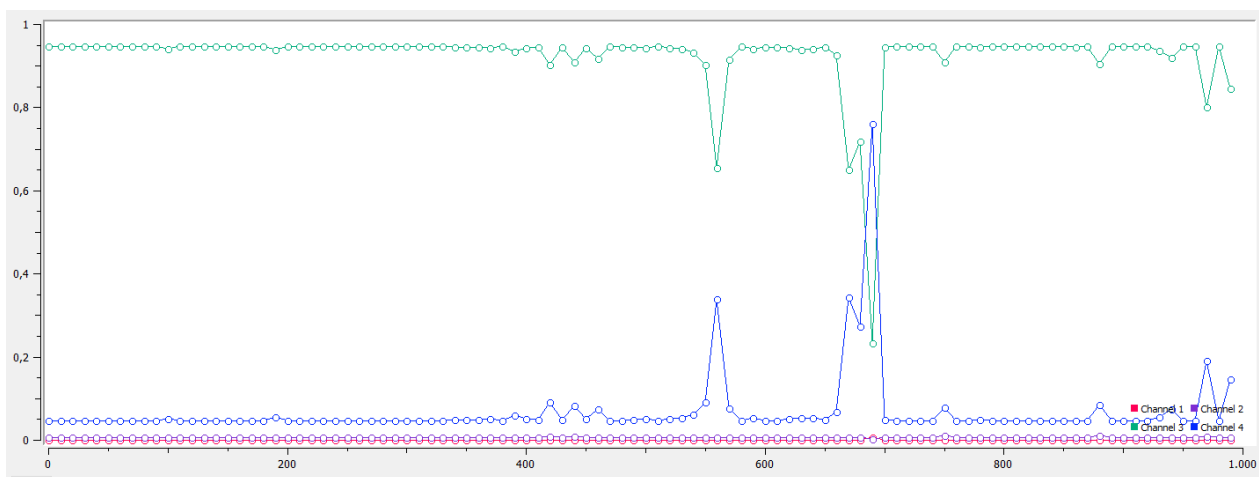


Figura 5.19 Resultado con mano hacia la izquierda.

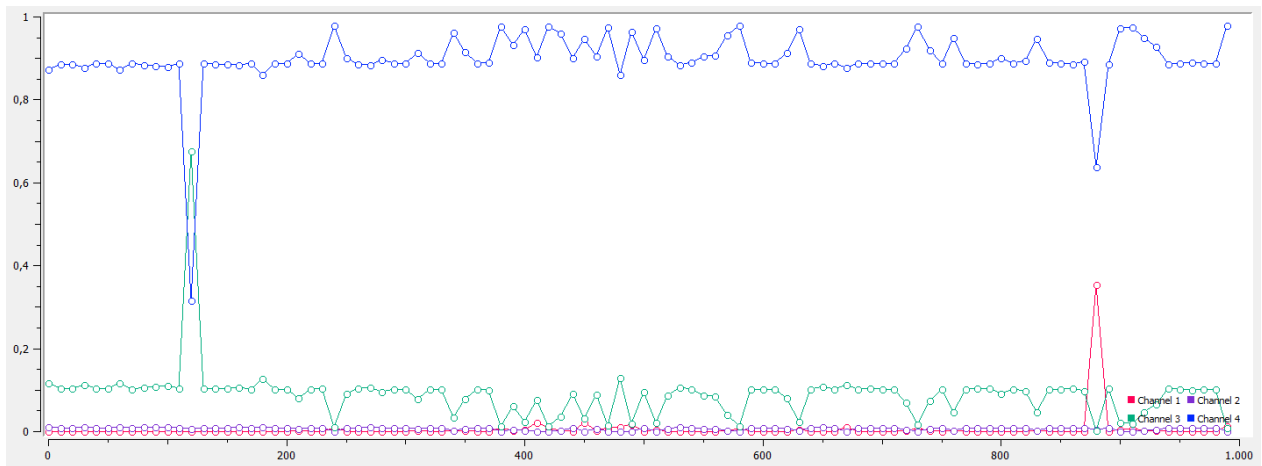


Figura 5.20 Resultado con mano abierta.

Cada gesto es representado con una señal de diferente color a lo largo del tiempo. En la ventana entera se muestran 100 resultados para 100 mediciones. El tiempo que toma en dar estos 100 resultados es de aproximadamente 13s lo cual implica que el ciclo entero de medición del ADC, extracción de características y paso por la red neuronal toma aproximadamente 130ms.

En los cuatro casos se puede apreciar una diferencia considerable entre el gesto que se predice y el resto, con pocas variaciones intermedias que podrían ser solucionadas amortiguando la interpretación de un gesto a partir de la probabilidad.

CAPITULO 6

PROGRAMA DE DEMOSTRACION

El programa de demostración sirve como ejemplo del uso del dispositivo de reconocimiento de gestos como elemento de control. Este programa recibe los datos que son enviados por el microcontrolador por serial y los utiliza como instrucciones.

Dado que la salida de la red neuronal es un conjunto de probabilidades, en vez de enviar dichas probabilidades por serial se implementó un último algoritmo en el microprocesador, que indica que gesto de los entrenados se está ejecutando. La regla utilizada es que el gesto realizado es el que tenga la mayor probabilidad pero solo si dicha probabilidad es mayor que la del resto de los gestos combinadas. Esto implica que en el caso que el gesto que la red neuronal predice tiene un porcentaje menor de acierto que desacierto no se actualiza el gesto que detecto en el ciclo anterior. Se considera la suma del resto de porcentajes como la probabilidad de desacierto ya que si cualquiera de los otros gestos es el correcto es una equivocación. Aunque este no es un método infalible, ayuda a eliminar ciertos errores.

Finalmente en caso que se determine que la probabilidad es suficientemente alta, se da como resultado una única variable que es un número entero que sirve para identificar los gestos. Con 4 gestos entrenados dicha variable “ID” puede valer 0, 1, 2 o 3, con cada valor sirviendo de identificación de cada gesto. Esta variable es la enviada por serial para ser interpretada por las aplicaciones.

El programa de demostración es una simulación 3D creada en el programa Unity3D, el cual es comúnmente utilizado para el desarrollo de videojuegos. La simulación consiste en 4 modelos 3d de una mano en diferentes posiciones como se observa en la figura 6.1. Dichos modelos fueron armados a partir de figuras geométricas existentes en Unity.



Figura 6.1 Modelos de la mano.

Cada uno de los modelos intenta ser un reflejo de los gestos entrenados que pueden ser realizados por el usuario. En las figuras 6.2, 6.3, 6.4, 6.5 se poder ver cada uno de los modelos por separado.



Figura 6.2 Mano en descanso.



Figura 6.3 Mano abierta.

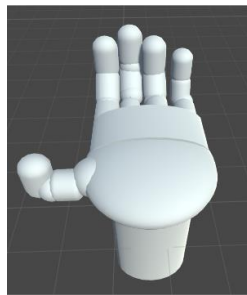


Figura 6.4 Mano hacia la izquierda.



Figura 6.5 Mano hacia la derecha.

Aunque estas no son animaciones propiamente dichas, es decir no es un modelo único que cambia moviéndose de una posición a otra, con estos 4 diseños es posible hacer la transición de un estado de la mano a otro simulando la animación. Para simular la transición, la señal de control lo que causa es el movimiento de la cámara de una posición a otra.

La recepción de la señal de control se configura de forma que cada vez que se actualicen los fotogramas del juego este chequea si hay un mensaje nuevo, leyendo un carácter. El número de identificación es enviado desde el microcontrolador como un carácter por lo tanto este es interpretado por su valor en código ASCII. Por esto al valor leído en Unity se le resta 48 para así obtener el número identificador del gesto realizado.

Con el identificador listo, simplemente se actualiza la posición de la cámara a la que corresponda para poder simular la animación de la mano. En las figuras 6.6, 6.7, 6.8 y 6.9 se ven exactamente las imágenes mostradas al usuario para cada gesto distinto.

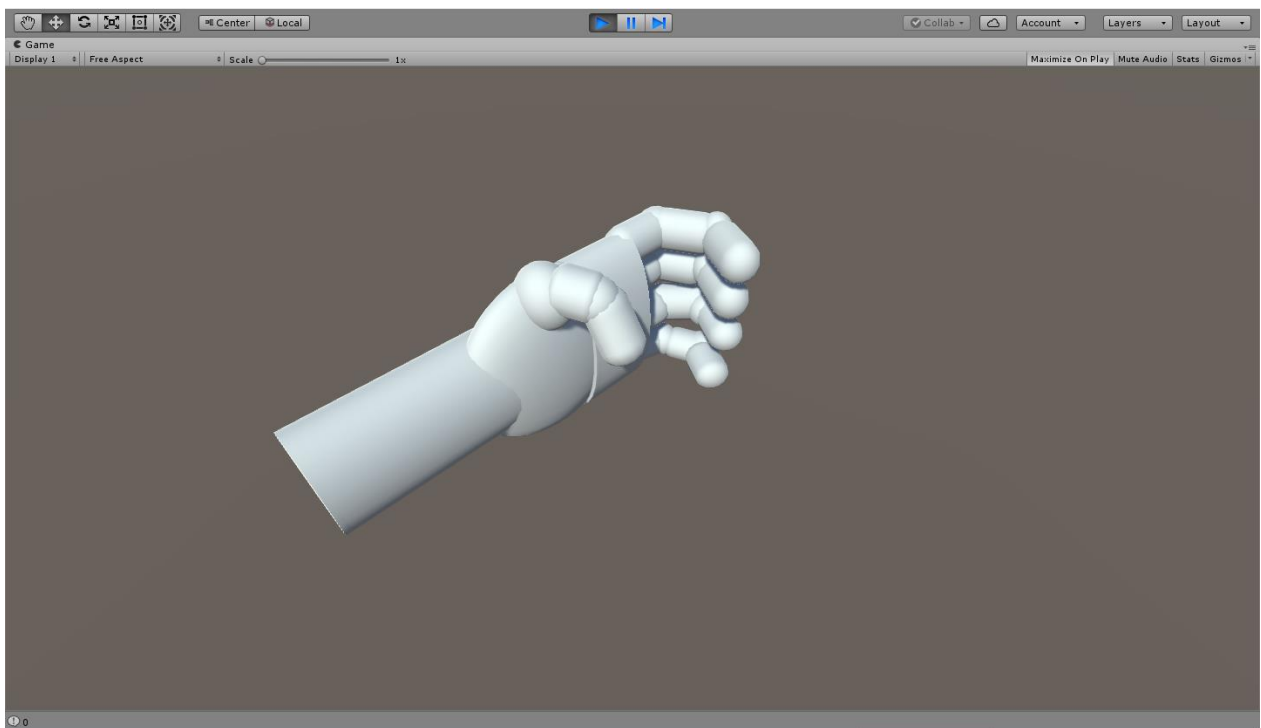


Figura 6.6 Mano hacia la descansando en programa en ejecución.

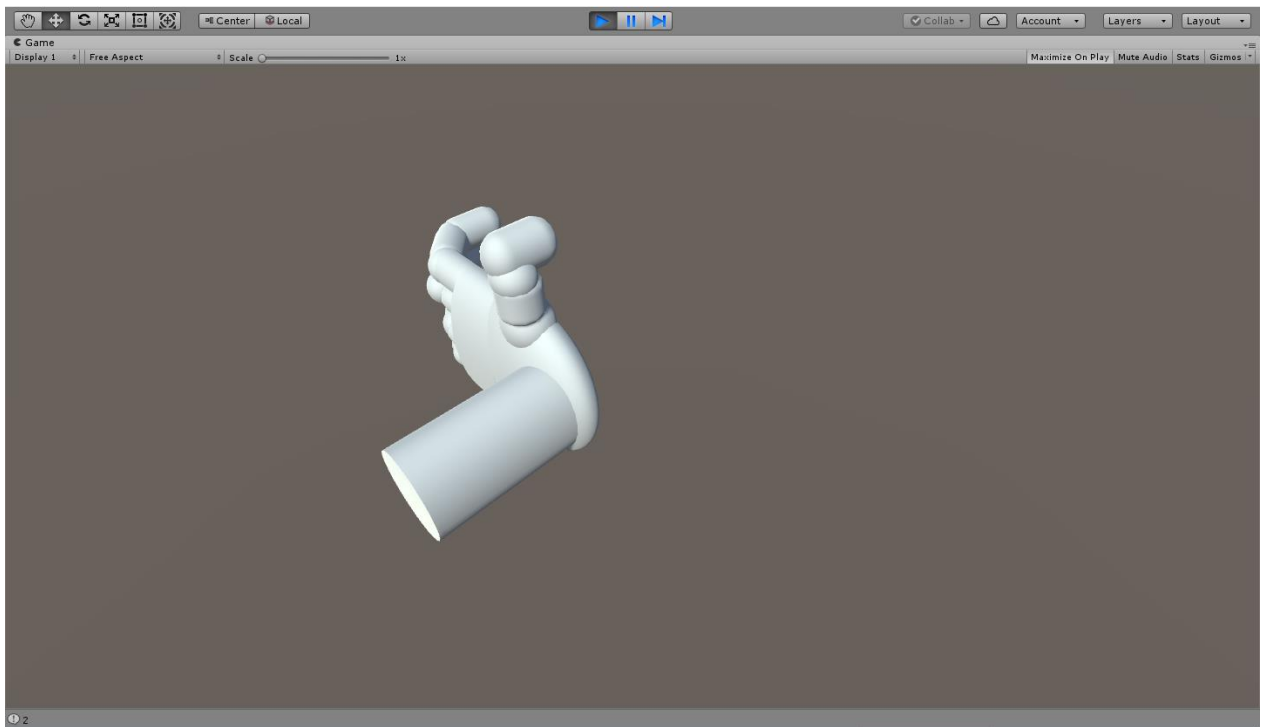


Figura 6.7 Mano hacia la izquierda en programa en ejecución.

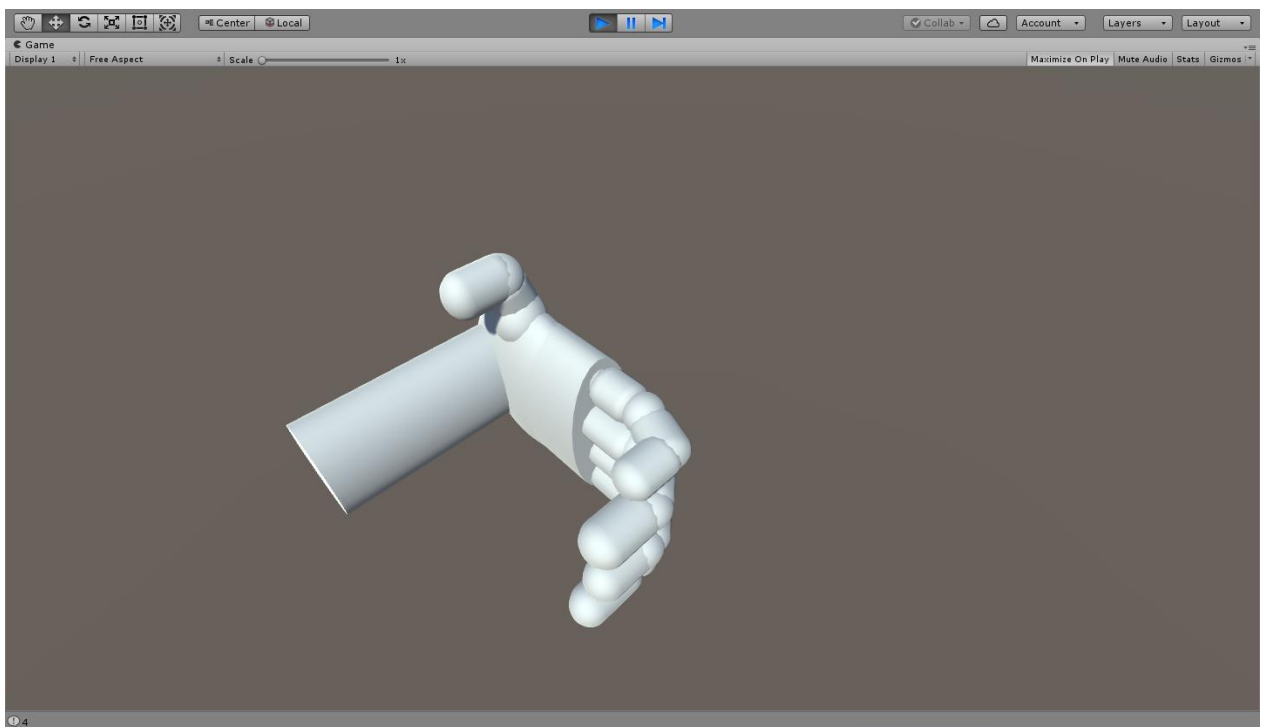


Figura 6.8 Mano hacia la derecha en programa en ejecución.

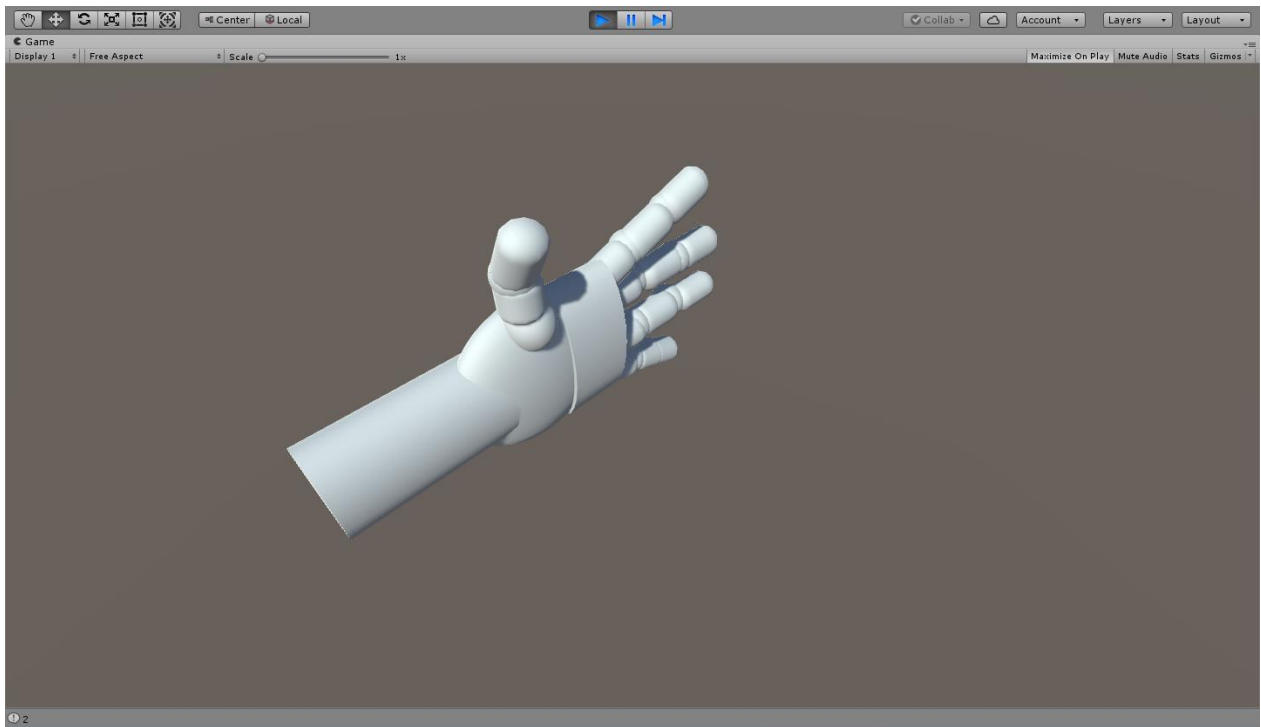


Figura 6.9 Mano abierta en programa en ejecución.

El cambio de la posición de la cámara en tiempo real logra el efecto de movimiento de la mano.

El funcionamiento de este programa además de demostrar el control logrado a partir de gestos, muestra que todo el trabajo puede descansar en el microprocesador dejando para la aplicación únicamente el uso de las señales de control como estas lo deseen. Esto es una muestra que su compatibilidad con otros sistemas se basa simplemente en la recepción y uso de la señal de control, sin requerir hacer diseños específicos para comprender la información que da la red neuronal.

CONCLUSIONES

Cada uno de los objetivos planteados fue cumplido. El resultado del proyecto fue el prototipo un dispositivo capaz de detectar un conjunto de gestos en tiempo real que son comunicados a un terminal y con dicha información se realiza el control de una aplicación.

Todas las etapas vistas por separadas cumplen con su cometido. El circuito de acondicionamiento logra llevar la señal EMG a un estado apto para ser estudiada y con los parámetros necesarios para poder ser adquirida por el procesador utilizado. Al final de esta etapa se mencionó que las interferencias introducidas por la situación del montaje (alimentación por fuente de computadora y montaje en la placa de prueba) podían ser fácilmente solucionadas, sin embargo algo importante es que a pesar de dicha situación, el sistema fue capaz de cumplir en cada una de las etapas subsiguientes, las cuales claramente dependen de la calidad de la primera sección y validez su funcionamiento.

El procesamiento de la señal que se lleva a cabo en el microprocesador logra cumplir con los puntos establecidos en el diseño. Se obtienen y guardan las señales de forma continua. Además se cumple que el sistema es fácilmente escalable, ya que como se puede ver en la estructura utilizada en el código del programa (apéndices A, B y C) la inclusión o intercambio de algoritmos sin perder compatibilidad con el resto del código es relativamente sencilla y permitiría también la inclusión de mediciones extra (más señales EMG).

La red neuronal artificial fue capaz de reconocer los gestos entrenados en tiempo real. La desventaja de la implementación actual viene por el entrenamiento de la misma. Aunque en teoría se puede entrenar cuantas veces se desee y actualizar la red en el microprocesador solo implica modificar el arreglo que contiene todos los pesos y el número de gestos entrenados, ciertamente el proceso actual es totalmente a mano, largo y tedioso. La continuación de este proyecto hasta llevarlo a un dispositivo totalmente móvil debe tener como objetivo número uno la automatización del proceso de entrenamiento.

Con el programa de demostración se cumple la simple tarea de demostrar la utilidad que se le puede dar al sistema y la ejecución del proceso entero en tiempo real.

La meta de este proyecto no fue simplemente lograr reconocer gestos sino que se buscó crear la base para un dispositivo totalmente móvil que pueda ser ubicado en el antebrazo (un

brazalete). Es por esto que en cada decisión de diseño se tuvo en mente las necesidades que tendría la futura evolución del proyecto, como por ejemplo la alimentación a partir de baterías, la estructuración del código para la fácil inclusión de algoritmos de extracción de características o incluso la necesidad que tendría el dispositivo de poseer una comunicación inalámbrica, la cual puede ser implementada con el microprocesador utilizado gracias a que este posee un periférico de comunicación Bluetooth. Estos y otros elementos aunque no fueron implementados en el prototipo, debían y fueron tomados en cuenta en el diseño.

RECOMENDACIONES

Hay una gran cantidad de elementos que se deberían tomar en cuenta no solo para mejorar el diseño actual sino para la evolución del prototipo hasta un dispositivo que sea un producto terminado. Algunos de estos ya han sido mencionados en el texto.

El elemento primordial a mejorar es el entrenamiento. El método actual sirvió para probar el funcionamiento de todo el sistema pero para un dispositivo finalizado es necesario el desarrollo de un programa que resulte en un sistema automatizado. Dicho programa comunicaría al microprocesador cuando cambiar entre los estados de entrenamiento en el que se enviaría constantemente las características y el estado de ejecución donde estaría corriendo la red y enviando los resultados. En la etapa de entrenamiento automáticamente se ordenarían los datos en diferentes gestos, se ejecutaría el algoritmo de entrenamiento y se enviarían al microprocesador todos los valores que se necesitan para armar la red. Todo esto con una interfaz sencilla que permita a cualquier usuario entrenar el dispositivo.

En segunda instancia, es necesario llevar el circuito a una tarjeta, implementar la alimentación por baterías, la comunicación inalámbrica, realizar el cambio a electrodos secos, la inclusión de más sistemas de adquisición y empacar todos los elementos en un brazalete que sea cómodo para el usuario. Además se podría utilizar el acelerómetro que posee el microprocesador lo que permitiría que las aplicaciones utilizaran el movimiento del brazo como otro elemento de control.

En el apartado del procesamiento de las señales, sería beneficioso cuantificar la influencia de cada característica en la red neuronal para así determinar qué características vale la pena mantener, cuales quitar y que alternativas existen que puedan mejorar el proceso tanto para el reconocimiento de más gestos como para la velocidad de cálculo.

Por último pero no menos importante, se debe considerar las aplicaciones en las cuales este sistema se acopla mejor y brinda la utilidad que no es posible obtener con otros sistemas de control. Existen ejemplos claros de sus potenciales usos como sistemas de control de prótesis donde este diseño permitiría expandir el rango de acciones posibles de la mano; videojuegos donde se podría simular la interacción del usuario con el espacio que lo rodea en un juego de realidad virtual; control de reproducción multimedia en teléfonos, computadoras; controlar

sistemas de domótica tanto para manejar fácilmente distintos aparatos en el hogar como automatizar tareas en oficinas.

REFERENCIAS

- [1] Microsoft, «Kinect for Windows,» [En línea]. Available: <https://developer.microsoft.com/en-us/windows/kinect>. [Último acceso: 1 Septiembre 2018].
- [2] LEAP MOTION, INC., «Leap Motion,» [En línea]. Available: <https://www.leapmotion.com/>. [Último acceso: 1 Septiembre 2018].
- [3] Thalmic Labs Inc., «Myo Armband,» [En línea]. Available: <https://www.myo.com/>. [Último acceso: 1 Septiembre 2018].
- [4] P. Konrad, «The ABC of EMG - A Practical Introduction to Kinesiological Electromyography,» Noraxon U.S.A, Inc. , Scottsdale, Arizona, 2006.
- [5] Wikipedia the free encyclopedia, «Machine Learning,» [En línea]. Available: https://en.wikipedia.org/wiki/Machine_learning. [Último acceso: 29 Agosto 2018].
- [6] G. Hinton, «Neural Networks for Machine Learning,» [En línea]. Available: <https://www.coursera.org/learn/neural-networks>. [Último acceso: 20 Septiembre 2018].
- [7] The MathWorks, Inc., «Linear Neural Networks,» [En línea]. Available: <https://la.mathworks.com/help/deeplearning/ug/linear-neural-networks.html>. [Último acceso: 20 Septiembre 2018].
- [8] The Mathworks, Inc., «Neural network performance,» [En línea]. Available: <https://la.mathworks.com/help/nnet/ref/crossentropy.html>. [Último acceso: 29 Agosto 2018].
- [9] Wikipedia the free encyclopedia, «Cross entropy,» [En línea]. Available: https://en.wikipedia.org/wiki/Cross_entropy. [Último acceso: 20 Septiembre 2018].
- [10] J. McCaffrey, «Neural Network Cross Entropy Error,» Visual Studio Magazine, 22 Abril 2014. [En línea]. Available: <https://visualstudiomagazine.com/articles/2014/04/01/neural-network-cross-entropy-error.aspx>. [Último acceso: 20 Septiembre 2018].
- [11] Wikipedia the free enciclopedia, «Propagacion hacia atras,» [En línea]. Available: https://es.wikipedia.org/wiki/Propagaci%C3%B3n_hacia_atr%C3%A1s. [Último acceso: 29 Agosto 2018].
- [12] Unity Technologies, «Unity3D,» 2018. [En línea]. Available: <https://unity3d.com/unity>. [Último acceso: 20 Septiembre 2018].
- [13] G. Herrera, «Acondicionamiento de la señal EMG,» de *Diseño conceptual del sistema de control para una prótesis de mano*, Caracas, 2017, pp. 43-76.
- [14] G. Urbina, «Acondicionamiento de la señal electromiográfica,» de *Diseño de dispositivo de control por gestos*, Caracas, 2017, pp. 34-57.

- [15] B. S. Day, «Important Factors in Surface EMG Measurement,» *Measurement*, pp. 1-17, 2002.
- [16] A. Albulbul, «Evaluating Major Electrode Types for Idle Biological Signal Measurements for Modern Medical Technology,» *Bioengineering*, vol. 3, nº 3, p. 20, agosto 2016.
- [17] Covidien (UK) Commercial Ltd, *Kendall™ ECG Electrodes Product Data Sheet*, vol. 44, Gosport Hampshire, 2008, p. 1.
- [18] C. J. De Luca, «The use of surface electromyography in biomechanics,» *Journal of Applied Biomechanics*, vol. 13, nº 2, pp. 135-163, 1997.
- [19] Burr-Brown Corporation, *INA121: FET-Input, Low Power INSTRUMENTATION AMPLIFIER Data Sheet*, 800 ed., vol. 6133, Tucson, 1998, pp. 1-12.
- [20] M. Tilal, «Classic filters,» [En línea]. Available: <http://electronics-2.weebly.com/uploads/1/3/0/5/13056901/classicfilters.pdf>. [Último acceso: 29 Agosto 2018].
- [21] Wikipedia the free encyclopedia, «Sallen Key Topology,» [En línea]. Available: https://en.wikipedia.org/wiki/Sallen%E2%80%933Key_topology. [Último acceso: 31 Agosto 2018].
- [22] Maxim Integrated Products, *MAX7480 8th-order, lowpass, Butterworth, switched-capacitor filter Data Sheet*, Sunnyvale, 1999, pp. 1-8.
- [23] NXP Semiconductors, *K22F Sub-Family Reference Manual*, 4 ed., 2016.
- [24] Texas Instruments Incorporated, *TI Precision Full-Wave Rectifier*, 2013, pp. 1-21.
- [25] STMicroelectronics, *L78 series - Positive voltage regulator Datasheet*, 34 ed., 2016, pp. 1-58.
- [26] STMicroelectronics, *L79 Series - Negative voltage regulators Datasheet*, vol. 23, 2017, pp. 1-27.
- [27] Arm Limited, «CMSIS DSP Software Library,» 22 Febrero 2018. [En línea]. Available: <https://www.keil.com/pack/doc/CMSIS/DSP/html/index.html>. [Último acceso: 29 agosto 2018].
- [28] A. Furui, H. Hayashi, Y. Kurita y T. Tsuji, «Variance distribution analysis of surface EMG signals based on marginal maximum likelihood estimation,» *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pp. 2514-2517, 2017.
- [29] D. Tkach, H. Huang y T. A. Kuiken, «Study of stability of time-domain features for electromyographic pattern recognition,» *Journal of NeuroEngineering and Rehabilitation*, vol. 7, nº 1, pp. 1-13, 2010.

- [30] The MathWorks, Inc., «Scaled conjugate gradient backpropagation,» [En línea]. Available: <https://la.mathworks.com/help/nnet/ref/trainscg.html>. [Último acceso: 29 Agosto 2018].
- [31] «Serial Plot,» [En línea]. Available: serialplot.com. [Último acceso: 29 Agosto 2018].
- [32] The MathWorks, Inc., «Hyperbolic tangent sigmoid transfer function,» [En línea]. Available: <https://la.mathworks.com/help/nnet/ref/tansig.html>. [Último acceso: 29 Agosto 2018].
- [33] The MathWorks, Inc., «Soft max transfer function,» [En línea]. Available: <https://la.mathworks.com/help/nnet/ref/softmax.html>. [Último acceso: 29 Agosto 2018].

APENDICES

Los apéndices A, B y C contienen código en lenguaje C que corre en el microcontrolador, creado dentro del ambiente de desarrollo MCUExpresso de NXP.

Apéndice A: Código para extracción de características.

En los archivos “Features_Calculator.h” y “Features_Calculator.c” están implementadas las funciones que extraen las características a partir del arreglo que contiene la señal EMG discretizada. La función principal “Features()” es la llamada en el programa principal “Hand_Análisis.c” (apéndice C) la cual ejecuta el resto de las funciones individuales para cada característica.

Para guardar los resultados se creó una estructura “Feat” formada por 6 elementos numéricos que guardan las 5 características y uno extra que se usa para ir guardando cada una de las mediciones del ADC pero que es solo usada para mantener temporalmente la información en el ciclo de adquisición de las 1000 mediciones.

Además aquí se definen 3 constantes globales que se usan en los diferentes apartados del código. El tamaño de la ventana el cual son 1000 medidas, el número de gestos y el número de neuronas en la capa oculta.

Features_Calculator.h:

```

1.  /*
2.   * Features_Calculator.h
3.   *
4.   * Created on: Apr 19, 2018
5.   * Author: linke
6.   */
7.
8.  #ifndef FEATURES_CALCULATOR_H_
9.  #define FEATURES_CALCULATOR_H_
10.
11. #define size 1000 //Number of window measures.
12. #define NG 4 //Number of Gestures.
13. #define HL 10 //Number of neurons in hidden layer.
14.
15. typedef struct Feat {

```

```

16.     float VOLT;           //ADC value transformed. Isn't a feature.
17.     float MEAN;           //Mean value of the measures window.
18.     //float IEMG;         //Cumulative value of EMG measures window. Unused
19.     float VAR;            //Variance.
20.     float WL;             //Waveform Length.
21.     float SSC;            //Samples sign count.
22.     float WAMP;           //Willison Amplitude.
23. }Feat;
24.
25. /*Calculates all features for the passed signal.*/
26. void Features (float *measures, float *abs_measures, float *mmo, float *mpo, Feat *result);
27.
28. /*MEAN*/
29. void Mean_Value ( float *abs_output, float *result);
30.
31. /*IEMG - Unused*/
32. void IEMG_calculator (float *abs_output, float *ones, float *result);
33.
34. /*Variance*/
35. void Variance (float *output, float *result);
36.
37. /*WaveformLength*/
38. void WaveformL (float *output, float *out_minus, float *difout, float *result);
39.
40. /*Willison Amplitude*/
41. void WillisonAmp (float*output, float *out_plus, float *result);
42.
43.
44. #endif /* FEATURES_CALCULATOR_H */

```

Features_Calculator.c

```

1.  /*
2.   * Features_Calculator.c
3.   *
4.   *   Created on: Apr 19, 2018
5.   *   Author: linke
6.   */
7.
8.  #include <stdio.h>
9.  #include "board.h"
10. #include "MK22F51212.h"
11. #include "Features_Calculator.h"
12. #include "arm_math.h"
13.
14. /*Auxiliary elements
15.  * ones, array to be filled with ones.
16.  * difout array to save the result of subtraction.
17.  * abs_difout array to save the absolute value of difout.
18.  * fx variables to save the features.
19.  */
20. float ones[size]={0.0f};
21. float difout[size]={0.0f};
22. float abs_difout[size]={0.0f};
23. float f1, f2, f3, f4, f5;
24.
25.
26. /*Main function that calls all the other specific features functions
27.  * and saves the results where it should.

```

```

28. */
29.
30. void Features (float *measures, float *abs_measures, float *mmo, float *mpo, Feat *result)
31. {
32.     //if the aux array hasn't been filled, fill it.
33.     if((ones[0]!=1.0f))
34.     {
35.         arm_fill_f32 (1.0f, &ones[0], size);
36.     }
37.
38.     //Get the absolute value of the measures array
39.     arm_abs_f32(measures, abs_measures, size);
40.
41.     Mean_Value (abs_measures, &f1);
42.     //IEMG_calculator (measures, &ones[0], &f2); //Unused, redundant with MEAN.
43.     Variance (measures, &f3);
44.     WaveformL(measures, mmo, ones, &f4);
45.     WillisonAmp(measures, mpo, &f5);
46.
47.     result->MEAN=f1;
48.     //result->IEMG=f2;
49.     result->WL=f3;
50.     result->VAR=f4;
51.     result->WAMP=f5;
52. }
53.
54. /*Mean IEMG value*/
55. void Mean_Value ( float *abs_output, float *result)
56. {
57.     //Calculate the mean value of the last N measures
58.     arm_mean_f32(abs_output, size, result);
59. }
60.
61. /*IEMG - Unused*/
62. void IEMG_calculator (float *abs_output, float *ones, float *result)
63. {
64.     arm_dot_prod_f32(abs_output, ones, size, result);
65. }
66.
67. /*Variance*/
68. void Variance (float *output, float *result)
69. {
70.     arm_var_f32(output, size, result);
71. }
72.
73. /*WaveformLenght*/
74. void WaveformL (float *output, float *out_minus, float *ones, float *result)
75. {
76.     //Subtract measures-measures shifted to the left.
77.     arm_sub_f32(output, out_minus, &difout[0], size);
78.     //Absolute value.
79.     arm_abs_f32(&difout[0], &abs_difout[0], size);
80.     //Sum of all array items.
81.     arm_dot_prod_f32(&abs_difout[0], ones, size, result);
82. }
83.
84. /*Willison Amplitude*/
85. void WillisonAmp(float*output, float *out_plus, float *result)
86. {
87.     //Subtract measures-measures shifted to the right.
88.     arm_sub_f32(output, out_plus, &difout[0], size);
89.

```



```
90.    //Absolute value.
91.    arm_abs_f32(&difout[0], &abs_difout[0], size);
92.
93.    //The result its basically a counter so it has to be reset on every run.
94.    *result=0;
95.
96.    //Checks if the difference its greater than the threshold.
97.    for(int m=0;m<size;m++)
98.    {
99.        if(abs_difout[m]>=0.01f) (*result)++;
100.    }
101. }
```

Apéndice B: Código de funciones de la red neuronal.

Los archivos “Functions.c” y “Functions.h” poseen las definiciones de las funciones de transferencia y la función que identifica el gesto realizado.

Las funciones de transferencia son tansig() y softmax() en las cuales se recrea las ecuaciones 5.1 y 5.2 respectivamente.

La función Gesture() determina el gesto realizado a partir de la salida de la red neuronal y devuelve un entero que identifica los gestos.

Functions.h

```

1.  /*
2.   * Functions.h
3.   *
4.   *   Created on: Apr 28, 2018
5.   *       Author: linke
6.   */
7.
8.  #ifndef FUNCTIONS_H_
9.  #define FUNCTIONS_H_
10.
11. //Tansig transfer function.
12. void tansig (float A[], int s);
13.
14. //Softmax transfer function.
15. void softmax(float A[], int s);
16.
17. //Determines which gesture was made.
18. int Gesture (float A[]);
19.
20. #endif /* FUNCTIONS_H_ */

```

Functions.c

```

1.  /*
2.   * Functions.c
3.   *
4.   *   Created on: Apr 28, 2018
5.   *       Author: linke
6.   */
7.  #include <stdio.h>
8.  #include "board.h"
9.  #include "MK22F51212.h"
10. #include "Functions.h"
11. #include "Features_Calculator.h"
12. #include "arm_math.h"
13. #include "math.h"
14.
15. float one[NG];
16.
17. //Tansig transfer function.
18. void tansig (float A[],int s)
19. {
20.     int i;

```

```

21.     float b;
22.     float c;
23.
24.     /*
25.      * Im not giving a math class, look for the math expression
26.      * and this is a recreation of it.
27.      * https://la.mathworks.com/help/nnet/ref/tansig.html
28.      */
29.     for(i=0;i<(s);i++)
30.     {
31.         b=-2*A[i];
32.         c=expf(b);
33.         A[i] = (2/(1+c))-1;
34.     }
35.
36. }
37.
38. //SoftMax transfer function.
39. void softmax(float A[], int s)
40. {
41.     int i;
42.     float b;
43.     float c=0;
44.
45.     /*
46.      * Im not giving a math class, look for the math expression
47.      * and this is a recreation of it.
48.      * https://la.mathworks.com/help/nnet/ref/softmax.html
49.      */
50.     for(i=0;i<(s);i++)
51.     {
52.         b=expf(A[i]);
53.         A[i] = b;
54.         c=c+b;
55.     }
56.     for(i=0;i<(s);i++)
57.     {
58.         A[i] = A[i]/c;
59.     }
60. }
61.
62. /*
63.  * Determines which gesture was made.
64.  * The rule is that if the probability of one gesture it's greater than
65.  * the rest combined, that's the gesture.
66.  */
67. int Gesture (float A[])
68. {
69.     int result;
70.     float a=0;
71.
72.     //if the aux array hasn't been filled, fill it.
73.     if((one[0]!=1.0f))
74.     {
75.         arm_fill_f32 (1.0f, &one[0], NG);
76.     }
77.
78.     //Sum of all array items done with dot product.
79.     arm_dot_prod_f32(&A[0], &one[0], NG, &a);
80.
81.     for(int i=0; i<=NG; i++)
82.     {
83.         //Checks if the i item is greater than the rest combined.

```

```
84.         if(A[i]>(a-A[i]))
85.         {
86.             result=i;
87.             break;
88.         }
89.     }
90.
91.     //Returns the index of the gesture, it starts at 0 not 1!
92.     return result;
93. }
```

Apéndice C: Código principal del dispositivo.

Este archivo contiene todo el proceso ejecutado en el microprocesador. Esto incluye:

- Se genera la señal de control para el filtro pasa bajo.
- Tomar medidas del ADC.
- Guardar datos en los arreglos.
- Llamar a la función de extracción de características descrita en “Features_Calculator.c” (apéndice A).
- Guardar las características en la matriz (vector) correspondiente.
- Realizar las multiplicaciones de matrices correspondientes a la red neuronal.
- Ejecutar las funciones de transferencia y la función que determina que gesto fue realizado a partir de la salida de la red neuronal descritas en “Functions.c” (apéndice B).
- Enviar por serial el identificador del gesto.

Además se definen todos los arreglos donde se guardan las medidas, pesos de las conexiones, características, etc... Y se definen con ellas las matrices.

Hand_Analisis.c

```

1.  /*
2.   * Copyright 2016-2018 NXP Semiconductor, Inc.
3.   * All rights reserved.
4.   *
5.   * Redistribution and use in source and binary forms, with or without modification,
6.   * are permitted provided that the following conditions are met:
7.   *
8.   * o Redistributions of source code must retain the above copyright notice, this list
9.   *   of conditions and the following disclaimer.
10.  *
11.  * o Redistributions in binary form must reproduce the above copyright notice, this
12.  *   list of conditions and the following disclaimer in the documentation and/or
13.  *   other materials provided with the distribution.
14.  *
15.  * o Neither the name of NXP Semiconductor, Inc. nor the names of its
16.  *   contributors may be used to endorse or promote products derived from this
17.  *   software without specific prior written permission.
18.  *
19.  * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
20.  * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
21.  * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
22.  * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
23.  * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
24.  * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
25.  * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
26.  * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
27.  * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS

```

```

28.  * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
29.  */
30.
31. /**
32.  * @file    Hand_Analisis.c
33.  * @brief   Application entry point.
34.  */
35. #include <stdio.h>
36. #include "board.h"
37. #include "peripherals.h"
38. #include "pin_mux.h"
39. #include "clock_config.h"
40. #include "MK22F51212.h"
41. #include "fsl_debug_console.h"
42. #include "arm_math.h"
43. #include "fsl_ftm.h"
44. #include "Features_Calculator.h"
45. #include "Functions.h"
46.
47.
48. /*ADC Constants*/
49. #define ADC1_Start ADC_Interrupt_Handler
50. #define ADC2_Start ADC2_Interrupt_Handler
51. #define Vref 3.300
52. #define bits 65536.0
53.
54. /*
55.  * These constants are the number of window measures, number of
56.  * trained gestures and number of neurons in hidden layer.
57.  * They are defined on Features_Calculator.h.
58.  * If the number of gestures changes, update the value.
59.  */
60. #define N size
61. #define G NG
62. #define Ne HL
63.
64. //Variable to save Channel 1 measure value.
65. volatile uint32_t valor1;
66.
67. //Variable to save Channel 2 measure value.
68. volatile uint32_t valor2;
69.
70. //Flags for ADC measures
71. volatile bool ADC_Flag1 = false;
72. volatile bool ADC_Flag2 = false;
73.
74.
75. //Initialize the ADCs
76. void ADC1_Start(void){
77.     ADC_Flag1 = true;
78.     valor1 = ADC16_GetChannelConversionValue(ADC_1_PERIPHERAL,0u);
79. }
80. void ADC2_Start(void){
81.     ADC_Flag2 = true;
82.     valor2 = ADC16_GetChannelConversionValue(ADC_2_PERIPHERAL,0u);
83. }
84.
85. /*
86.  * Neural Network Data
87.  */
88. //Array to save all the inputs (features) plus the bias.
89. float input[Ne+1];
90.

```

```

91. /*
92.  * BW1 & BW2 are the NN weight arrays.
93.  * These are the arrays that have to be refreshed when the NN it's retrained.
94.  * They are "converted" to matrix later.
95.  * */
96. float BW1[] = {0.387013, -3.7217, -0.105211, 5.82028, 0.0086538, -0.254089, 5.01615, -
    0.00977728, -9.2619, -0.0363502, -0.27898, -2.50321, 2.69253, -
    0.220488, 18.1042, 0.191931, 0.0132072, -0.771766, -0.230785, 8.11876, 0.248018, -0.0145511, -
    1.57895, -2.37682, 0.419658, 20.8802, 0.0355196, 0.711133, 0.237763, -
    0.0628312, 12.2687, 0.0226422, -0.0326549, 4.51197, -
    2.06664, 0.0976717, 22.2227, 0.065327, 0.0229626, -
    6.30569, 0.0540829, 4.5099, 0.037585, 0.0392922, -0.610813, -7.31294, 0.56197, 22.1472, -
    0.10213, 0.590431, 2.35736, 0.410547, -11.4643, -0.0679317, -0.0324921, -1.65986, -
    2.60551, 0.0487882, -0.854315, 0.246824, 0.607218, 1.58287, 0.0515319, -
    13.9679, 0.138308, 0.723637, -2.6013, 0.718456, 0.0670275, -15.363, -0.0229018, -
    0.148285, 0.426316, 0.00704314, -7.39401, -0.0368661, -0.179606, -2.51109, -1.29187, -
    0.225269, 34.8558, 0.283184, 0.0282652, 2.77611, -
    0.504365, 10.5615, 0.209483, 0.059099, 0.37877, 1.25601, -0.0367713, -
    26.1169, 0.102869, 0.00856349, 4.68644, -0.219841, -13.3948, 0.1744, -0.135423, -
    1.86717, 7.12173, -0.145154, -5.56873, -0.108048, 0.831623, 3.07379, -0.178763, -10.2247, -
    0.0128999, -0.24963};
97. float BW2[] = {0.245772, 0.0403525, -1.00947, -0.891614, -0.260954, -0.0671591, -0.118392, -
    0.0357653, -1.58065, -0.314836, -0.654904, 0.0503466, 0.592063, -0.00178819, -0.629867, -
    0.851089, -1.1726, 0.979834, -0.839649, 1.75353, -0.573732, -
    0.457148, 0.542396, 0.760147, 0.168547, 0.859029, -0.600638, 0.0801297, 0.753935, -
    0.396098, 1.17579, 0.533974, 1.14622, 0.17879, -0.813973, -1.19739, 0.750112, -
    0.405469, 0.67901, 0.593883, 0.0430763, -0.196537, -0.689156, 0.792834};
98.
99. /*
100.  * These arrays are for the neurons.
101.  * Layer1 its the hidden layer
102.  * Layer12 its the same as 1 but with the extra 1 of the Bias
103.  * and Layer 2 it's the output layer.
104.  * */
105. float Layer1[Ne];
106. float Layer12[Ne+1];
107. float Layer2[G];
108.
109. /*Gesture identification number.
110.  *Will be used to inform the application the gesture
111.  */
112. int ID;
113.
114.
115. /*
116.  * @brief Application entry point.
117.  */
118. int main(void) {
119.
120.     /* Init board hardware. */
121.     BOARD_InitBootPins();
122.     BOARD_InitBootClocks();
123.     BOARD_InitBootPeripherals();
124.     /* Init FSL debug console. */
125.     BOARD_InitDebugConsole();
126.
127.     /*1st channel data arrays*/
128.
129.     //Array of last measures.
130.     float samples[N]={0.0f};
131.     float *measures = &samples[0];
132.
133.     //Absolute value of the last measures.

```

```

134.         float abs_samples[N]={0.0f};
135.         float *abs_measures = &abs_samples[0];
136.
137.
138.         //Array to save last measures shifted one.
139.         float samples_minus_one[N]={0.0f};
140.         float *SMO=&samples_minus_one[0];
141.
142.         //Array to save last measures shifted one.
143.         float samples_plus_one[N]={0.0f};
144.         float *SPO=&samples_plus_one[0];
145.
146.
147.         /*2nd channel data arrays*/
148.
149.         //Array of last measures.
150.         float samples2[N]={0.0f};
151.         float *measures2 = &samples2[0];
152.
153.         //Absolute value of the last measures.
154.         float abs_samples2[N]={0.0f};
155.         float *abs_measures2 = &abs_samples2[0];
156.
157.         //Array of last measures, shifted one - right.
158.         float samples_minus_one2[N]={0.0f};
159.         float *SMO2=&samples_minus_one2[0];
160.
161.         //Array of last measures, shifted one - left.
162.         float samples_plus_one2[N]={0.0f};
163.         float *SPO2=&samples_plus_one2[0];
164.
165.
166.         /*
167.          * Used to keep track of how many measures are left to fill the window
168.          * */
169.         int ADC_Cycles=(N-1);
170.
171.         /*
172.          * Structures to save results (all 5 features)
173.          * "Feat" structure its defined on Features_Calculator.C
174.          * If more channels are measured, more structures need to be initialized.
175.          * */
176.         //Channel 1
177.         Feat results;
178.         //Channel 2
179.         Feat results2;
180.
181.
182.         /*
183.          * NN Matrices
184.          * */
185.         //Matrix with the weights for Layer 1
186.         arm_matrix_instance_f32 NN1 = {Ne, (Ne+1), &BW1[0]};
187.
188.         //Vector with the inputs (features)
189.         arm_matrix_instance_f32 X = {(Ne+1),1, &input[0]};
190.
191.         //Matrix with the weights for Layer 2
192.         arm_matrix_instance_f32 NN2 = {G, (Ne+1), &BW2[0]};
193.
194.         //Hidden layer vector
195.         arm_matrix_instance_f32 L1 = {Ne,1,&Layer1[0]};
196.

```



```

197.      //Hidden layer vector with bias
198.      arm_matrix_instance_f32 L12 = {(Ne+1),1,&Layer12[0]};
199.
200.      //Output layer vector
201.      arm_matrix_instance_f32 L2 = {G,1,&Layer2[0]};
202.
203.
204.      /*
205.       * Enter the main loop. May the force be with you...
206.       * */
207.
208.      while(1)
209.      {
210.          /*Get N measures from ADC*/
211.          while(ADC_Cycles>=0){
212.
213.              ADC_Flag1 = false;
214.              ADC_Flag2 = false;
215.
216.              ADC16_SetChannelConfig(ADC_1_PERIPHERAL,0U,&ADC_1_channelsConfig[0]
217.          );
218.              ADC16_SetChannelConfig(ADC_2_PERIPHERAL,0U,&ADC_2_channelsConfig[0]
219.          );
220.              /*Wait for the measures to finish*/
221.              while(!(ADC_Flag1&ADC_Flag2))
222.              {
223.              }
224.
225.              /*Transform the measure to volts*/
226.              results.VOLT=valor1*((Vref/bits));
227.              results2.VOLT=valor2*((Vref/bits));
228.
229.              /*Save the measure on the window array*/
230.              samples[ADC_Cycles]=results.VOLT;
231.              samples2[ADC_Cycles]=results2.VOLT;
232.
233.              /*
234.               * Filling the auxiliary arrays
235.               * these have the same values but shifted positions
236.               * */
237.              if (ADC_Cycles>0){
238.                  samples_minus_one[ADC_Cycles-1]=results.VOLT;
239.                  samples_minus_one2[ADC_Cycles-1]=results2.VOLT;
240.              }
241.              if (ADC_Cycles<(N-1)){
242.                  samples_plus_one[ADC_Cycles+1]=results.VOLT;
243.                  samples_plus_one2[ADC_Cycles+1]=results2.VOLT;
244.              }
245.
246.              ADC_Cycles=ADC_Cycles-1;
247.          }
248.
249.          //Reset the ADC cycle.
250.          ADC_Cycles=N-1;
251.
252.          //The auxiliary arrays have the last or first position on 0
253.          samples_minus_one[N-1]=0.0f;
254.          samples_plus_one[0]=0.0f;
255.          samples_minus_one2[N-1]=0.0f;
256.          samples_plus_one2[0]=0.0f;
257.

```

```

258.
259.
260.
261.
262.      /*
263.       * Call the Features function
264.       * It extracts the features
265.       * and returns a "result" structure filled
266.       * */
267.
268.      Features(measures, abs_measures, SMO, SPO, &results);
269.      Features(measures2, abs_measures2, SMO2, SPO2, &results2);
270.
271.      /**
272.       * Slope Sign Changes:
273.       * This feature is done here instead of Features_Calculator.c
274.       * like the rest of the features because it's harder to access
275.       * each member of the samples array using the pointers, so yeah
276.       * because i'm too lazy to do it the smart way...
277.       * Counts the times the slope sign changes passing a threshold.
278.       * Because the results are counters they have to be reset.
279.       * */
280.
281.      results.SSC=0;
282.      results2.SSC=0;
283.
284.      for(int n=0;n<N;n++)
285.      {
286.          //Channel 1
287.          if (((samples[n]-samples_minus_one[n])*(samples[n]-
samples_plus_one[n]))>0.00001f) (results.SSC)++;
288.          //Channel 2
289.          if (((samples2[n]-samples_minus_one2[n])*(samples2[n]-
samples_plus_one2[n]))>0.00001f) (results2.SSC)++;
290.      }
291.
292.      /*
293.       * Pass all the features to the corresponding array.
294.       * The "X" vector it's linked to "input[]" memory position
295.       * so this is filling the input vector.
296.       * input[0] is the first Bias.
297.       * If channels are added the rest of the features have to be saved.
298.       * */
299.      input[0] = 1.0f;          //Bias
300.      input[1] = results.MEAN;
301.      input[2] = results.VAR;
302.      input[3] = results.WL;
303.      input[4] = results.SSC;
304.      input[5] = results.WAMP;
305.      input[6] = results2.MEAN;
306.      input[7] = results2.VAR;
307.      input[8] = results2.WL;
308.      input[9] = results2.SSC;
309.      input[10] = results2.WAMP;
310.
311.      /*
312.       * (1st Weight matrix)*(Input matrix) = first layer
313.       * then pass the result to the "Tansig" transfer function and get
314.       * the Hidden Layer.
315.       * Tansig it's defined on Functions.c
316.       * */
317.      arm_mat_mult_f32 (&NN1,&X,&L1);

```

```

318.         tansig (Layer1, (sizeof(Layer1)/sizeof(Layer1[0])));
319.
320.         /*
321.          * Add the second Bias.
322.          * Basically copy the vector with the extra 1.
323.          * */
324.         for(int i=0;i<(sizeof(Layer1)/sizeof(Layer1[0]));i++)
325.         {
326.             Layer12[i+1]=Layer1[i];
327.         }
328.         Layer12[0]=1;
329.
330.
331.         /*
332.          * (2nd Weight matrix)*(Hidden layer matrix) = second layer
333.          * then pass the result to the "SoftMax" transfer function
334.          * and get the Output Layer.
335.          * SoftMax it's defined on Functions.c
336.          * */
337.         arm_mat_mult_f32 (&NN2,&L12,&L2);
338.         softmax(Layer2, (sizeof(Layer2)/sizeof(Layer2[0]))); //Functions.C
339.
340.         /*
341.          * We search which gesture was made based on which has the highest
342.          * probability.
343.          * The probability must be greater than the rest of probabilities c
344.          * ombined.
345.          * "Gesture()" is defined on Functions.c
346.          * Each gesture is represented by a number, ex:0,1,2,3 for 4 gestur
347.          * es.
348.          * */
349.         ID = Gesture(&Layer2[0]);
350.
351.         /*
352.          * Send all 10 features, used for training
353.          * */
354.         //PRINTF("%f, %f, %f, %f, %f, %f, %f, %f, %f, %f, %f\n",results.
355.         MEAN, results.VAR, results.WL, results.SSC, results.WAMP, results2.MEAN, results2.VAR,
356.         results2.WL, results2.SSC, results2.WAMP);
357.
358.         /*
359.          * Send features of 1st signal channel. Just for tests.
360.          * */
361.         //PRINTF("%f, %f, %f, %f, %f\n", results.MEAN, results.VAR, resul
362.         ts.WL, results.SSC, results.WAMP);
363.
364.         /*
365.          * Send features of 2nd signal channel. Just for tests.
366.          * */
367.         //PRINTF("%f, %f, %f, %f\n", results2.MEAN, results2.VAR, resul
368.         ts2.WL, results2.SSC, results2.WAMP);
369.
370.         /*
371.          * Send NN output. These are the probabilities.
372.          * */
373.         //PRINTF("%f, %f, %f, %f\n",Layer2[0],Layer2[1],Layer2[2],Layer2[3]
374.         );
375.
376.         /*
377.          * Send the gesture ID. IT'S BEEN 84 YEARS!!!!

```

```
372.          * */
373.          PRINTF("%d", (ID));
374.
375.      }
376.
377.      return 0 ;
378.  }
```

Apéndice D: Código para entrenamiento de la red neuronal

Este es el código utilizado para realizar el entrenamiento de la red, el mismo se ejecuta en Matlab. Inicialmente el código es generado al utilizar la interfaz de Matlab para el uso de redes neuronales sin embargo al ir al código base y hacer cambios es posible modificar el algoritmo de entrenamiento para leer los datos (las matrices de entrenamiento) a partir de los archivos “.CSV” además de guardar los resultados de la misma manera.

NetScript1.m

```

1. % Solve a Pattern Recognition Problem with a Neural Network
2. % Script generated by Neural Pattern Recognition app
3. % Created 24-Apr-2018 17:56:39
4. %
5. % This script assumes these variables are defined:
6. %
7. %   Features - input data.
8. %   Results - target data.
9. clear all
10. clc
11.
12. %Read the input
13.
14. x = transpose(csvread('todo3.csv'));
15. t = transpose(csvread('res2.csv'));
16.
17. % Choose a Training Function
18. % For a list of all training functions type: help nntrain
19. % 'trainlm' is usually fastest.
20. % 'trainbr' takes longer but may be better for challenging problems.
21. % 'trainscg' uses less memory. Suitable in low memory situations.
22. trainFcn = 'trainscg'; % Scaled conjugate gradient backpropagation.
23.
24. % Create a Pattern Recognition Network
25. hiddenLayerSize = 10;
26. net = patternnet(hiddenLayerSize, trainFcn);
27.
28. % Setup Division of Data for Training, Validation, Testing
29. net.divideParam.trainRatio = 70/100;
30. net.divideParam.valRatio = 15/100;
31. net.divideParam.testRatio = 15/100;
32. net.inputs{1}.processFcns = {};
33. %net.layers{1}.transferFcn = 'logsig';
34. %net.layers{2}.transferFcn = 'tansig';
35.
36.
37. % Train the Network
38. [net,tr] = train(net,x,t);
39.
40. % Test the Network
41. y = net(x);
42. e = gsubtract(t,y);
43. performance = perform(net,t,y)

```

```

44. tind = vec2ind(t);
45. yind = vec2ind(y);
46. percentErrors = sum(tind ~= yind)/numel(tind);
47.
48. % View the Network
49. view(net)
50.
51. % Plots
52. % Uncomment these lines to enable various plots.
53. %figure, plotperform(tr)
54. %figure, plottrainstate(tr)
55. %figure, ploterrhist(e)
56. %figure, plotconfusion(t,y)
57. %figure, plotroc(t,y)
58.
59. %Save the network on various matrices
60. W1=net.IW{1};
61. B1=net.b{1};
62. W2=net.LW{2};
63. B2=net.b{2};
64. BW1=[B1,W1];
65. BW2=[B2,W2];
66.
67. %Just a test of the transfer functions equations
68. %ya = net.IW{1} * x + net.b{1};
69. %L1 = W1 * x + B1;
70. %y1 = tansig(ya);
71. %L1 = 2./(1+exp(-2*L1))-1;
72. %r = net.LW{2} * y2 + net.b{2};
73. %L2 = W2 * L1 + B2;
74. %Results = softmax(r);
75. %Results3 = exp(L2)./sum(exp(L2));
76.
77. %Just to see which would be the results of passing the whole trainig
78. %input matrix through the trained NN
79. Results2 = sim(net,x);
80.
81. %save the weights on CSV
82. dlmwrite('BW1.csv', BW1, 'delimiter', ',', 'precision', 6);
83. dlmwrite('BW2.csv', BW2, 'delimiter', ',', 'precision', 6);

```

Apéndice E: Código del programa de ejemplo.

Este archivo realiza la lectura del puerto serial por parte del programa y controla la posición de la cámara a partir del gesto recibido. Escrito en lenguaje C#.

Cam_follow.cs

```

1. using System.Collections;
2. using System.Collections.Generic;
3. using UnityEngine;
4. using System.IO.Ports;
5.
6. public class Cam_follow : MonoBehaviour
7. {
8.     /*This variable saves Transform property of the object in this case the camera.
9.     * It includes position, rotation and scale.
10.    */
11.    private Transform tf;
12.
13.    /* Define the port.
14.    * Normally the program should scan the ports until
15.    * it finds the device but that's extra work.
16.    */
17.    public SerialPort serial = new SerialPort("\\\\.\\COM5", 115200);
18.
19.    /*Variable to save the gesture ID*/
20.    public int s_data;
21.
22.    /* Initialization
23.    * Open serial port, set timeout ( for security)
24.    * and assign the "object transform" to the variable.
25.    */
26.    void Start()
27.    {
28.        serial.Open();
29.        serial.ReadTimeout = 200;
30.        print(serial.IsOpen);
31.        tf = GetComponent<Transform>();
32.    }
33.
34.    /*Main code
35.    * Update is called once per frame but "FixedUpdate()"
36.    * can be set to a specific refresh rate if needed.
37.    * Checks if port is still open, send gesture ID to console (just for monitoring)
38.    * and decide the camera position depending on the serial input.
39.    * Because the ID is an ASCII symbol, we subtract 48 to get the real number.
40.    */
41.    void Update()
42.    {
43.
44.        if (!serial.IsOpen)
45.        {
46.            print(serial.IsOpen);
47.            serial.Open();
48.        }
49.
50.        s_data = ((serial.ReadChar())-48);
51.        print(s_data);
52.

```

```

53.      /*Change camera position depending on the gesture.*/
54.      switch (s_data)
55.      {
56.          /*Resting hand*/
57.          case 0:
58.              transform.position = new Vector3(-40, tf.position.y, tf.position.z);
59.              break;
60.
61.          /*Hand to the right*/
62.          case 1:
63.              transform.position = new Vector3(85, tf.position.y, tf.position.z);
64.              break;
65.          /*Hand to the left*/
66.          case 2:
67.              transform.position = new Vector3(45, tf.position.y, tf.position.z);
68.              break;
69.
70.          /*Open hand*/
71.          case 3:
72.              transform.position = new Vector3(5, tf.position.y, tf.position.z);
73.              break;
74.          default:
75.              break;
76.      }
77.
78.      /*
79.       * To test control with keyboard
80.       if (Input.GetKey(KeyCode.A))
81.       {
82.           transform.position = new Vector3(5, tf.position.y, tf.position.z);
83.       }
84.       if (Input.GetKey(KeyCode.S))
85.       {
86.           transform.position = new Vector3(45, tf.position.y, tf.position.z);
87.       }
88.       if (Input.GetKey(KeyCode.D))
89.       {
90.           transform.position = new Vector3(-40, tf.position.y, tf.position.z);
91.       }
92.       if (Input.GetKey(KeyCode.F))
93.       {
94.           transform.position = new Vector3(85, tf.position.y, tf.position.z);
95.       }
96.      */
97.  }
98. }

```