



Proyecto Final:

Desarrollo de un electromiógrafo aplicado al control de una mano biónica.

INTEGRANTES:

GUIDI, Juan Cruz - L.U: 16.546

DOCENTES:

Mg. FRIEDRICH, Guillermo
Ing. LAIUPPA, Adrián

Resumen

La electromiografía parte del mismo principio que la electrocardiografía, monitorear señales biológicas de voltaje reducido. Generalmente, ésta, requiere de equipos de coste elevado, siendo esta inaccesible en ciertos ámbitos de bajos recursos económicos.

Las principales complicaciones a la hora de reducir dichos costes se centran en encontrar plataformas de pequeño tamaño, alto rendimiento y bajo consumo energético que puedan capturar señales biológicas en tiempo real.

Se presenta un dispositivo de bajo coste que cumple con lo descripto anteriormente, haciendo uso de una Raspberry Pi Zero W, Para permitir el procesamiento de la señal y comunicación con la aplicación celular.

La meta final del dispositivo realizado es conseguir medir la actividad muscular, mostrarla en una aplicación en un teléfono celular y a partir de estos datos, controlar el movimiento de una mano electromecánica. Todo esto con un coste lo más reducido posible.

Índice general

| | |
|--|-----------|
| 1. Introducción | 1 |
| 1.1. Motivación | 2 |
| 1.2. Visión General | 4 |
| | |
| 2. Antecedentes | 5 |
| 2.1. Electromiografía | 6 |
| 2.1.1. Característica eléctrica | 6 |
| 2.1.2. Procedimiento | 7 |
| 2.1.3. Instrumental | 7 |
| 2.1.4. Limitaciones EMG superficial | 9 |
| | |
| 2.2. Prótesis | 10 |
| 2.2.1. Mano mecánica | 12 |
| 2.2.2. Manos electromecánicas | 12 |
| | |
| 2.3. Tecnologías | 14 |
| 2.3.1. Raspberry Pi Zero W | 14 |
| 2.3.2. ADS1294 | 15 |
| 2.3.3. MyoWare Muscle Sensor | 17 |
| 2.3.4. Bus I ² C | 20 |
| 2.3.5. Bluetooth de baja energía (BLE) | 22 |
| 2.3.6. NodeJS | 29 |
| 2.3.7. Swift | 30 |
| 2.3.8. Impresión 3D | 31 |
| | |
| 3. Criterios de diseño | 33 |
| 3.1. Raspbian | 34 |
| 3.2. Sensor muscular | 34 |
| 3.3. Programación | 35 |
| 3.3.1. Sobre la Raspberry | 35 |
| 3.3.2. Aplicación móvil | 35 |
| | |
| 3.4. Modelo 3D | 36 |
| 3.5. Modelo del sistema EMG | 38 |
| 3.6. Conclusiones de diseño | 39 |

| | |
|---|-----------|
| 4. Desarrollo | 40 |
| 4.1. Estructura del sistema | 41 |
| 4.1.1. Medición de la actividad muscular | 41 |
| 4.1.2. Comunicación entre dispositivos I ² C | 45 |
| 4.1.3. Procesamiento de datos | 46 |
| 4.1.4. Comunicación entre dispositivos BLE | 46 |
| 4.1.5. Aplicación iOS | 49 |
| 4.1.6. Construcción de la mano prostética | 50 |
| 4.1.7. Movimiento de la mano prostética | 52 |
| 4.2. Tiempo de trabajo y costos | 53 |
| 5. Resultados | 54 |
| 5.1. Sistema final | 55 |
| 5.2. Pasos a futuro y posibles mejoras | 55 |
| Referencias Bibliográficas | 58 |
| Anexo I - MyoWare Muscle Sensor | |
| Anexo II - Servidor GATT NodeJS | |
| Anexo III - Aplicación iOS | |

1. Introducción

En este trabajo se presenta el desarrollo de una aplicación para la realización de una electromiografía, que corre sobre el sistema operativo iOS, conectada por bluetooth con una Raspberry Pi Zero W (RPi)^[9] sobre la cual corre el software de muestreo y transmisión de los datos de la medición.

Además con estos datos muestreados sobre la RPi y la opción de configuración mediante la nombrada aplicación, se realiza el control de una mano electromecánica impresa en 3D.

Se exponen los casos de estudio que se tuvieron en cuenta para la elección de los componentes y los resultados obtenidos.

En esta primera parte se explica con detalle en que consiste este proyecto y la motivación para llevarlo a cabo y se exponen las razones de elección de la plataforma utilizada.

1.1. Motivación

Usar nuestros músculos para controlar y mover objetos es la forma más común en la que muchos de nosotros estamos acostumbrados a hacer las cosas. Sobre todo utilizar las manos para manipular objetos, controles y hasta para comunicarnos mediante señas.

Estos músculos son controlados por neuronas llamadas neuronas motoras, las cuales transmiten señales eléctricas que hacen que los músculos se contraigan. Para poder controlar cosas mediante estas señales eléctricas necesitamos de alguna manera convertirlas en datos que puedan ser utilizados por un programa o dispositivo. Esto es lo que hace la electromiografía (EMG) (un procedimiento de diagnóstico que se utiliza para evaluar la salud de los músculos y las neuronas que los controlan), convertir estas señales en gráficos, sonidos o valores numéricos que interpreta un especialista. Para lograrlo usa dispositivos diminutos denominados electrodos para transmitir o detectar señales eléctricas.

Normalmente, como es de suponer, estos equipos medicos son costosos, pero de una alta precisión, ya que su fin es detectar y diagnosticar enfermedades nerviosas o problemas musculares.

A fin de realizar el control de una mano impresa en 3D mediante la actividad muscular, se debe realizar un dispositivo capas de realizar una EMG.

Para el objetivo de este trabajo, no se necesita una precisión tan alta, pero en cambio si portabilidad y accesibilidad. En estos últimos dos puntos son donde se destacan las opciones elegidas para llevar a cabo el trabajo.

El bajo costo, bajo consumo y conectividad de la Raspberry Pi Zero W^[9] la hace ideal para tareas de este estilo, donde se necesite en un espacio reducido una capacidad de procesamiento alta.

Dicha conectividad es requerida para la comunicación con la aplicación celular en donde se visualizarían los datos y se podría configurar el movimiento de la mano.

Se utiliza una aplicación celular para evitar la creación de un dispositivo de mayor volumen y peso, además de la comodidad que supone la utilización de un dispositivo ya conocido.

En lo que respecta a manos ortopédicas, existen mecánicas, donde es un movimiento lo que genera el movimiento de la mano (generalmente abrir y cerrar), pero también existen electromecánicas, las cuales permiten mayor cantidad de movimientos, pero con un coste mucho más elevado.

Se busca en este caso, poder lograr una prótesis electromecánica de bajo coste gracias a la utilización de open-hardware y de la tecnología de impresión 3D.

1.2. Visión general

En esta sección se pretende ofrecer una idea general de la organización de este documento así como una visión de este proyecto y los temas cubiertos por la investigación realizada para llevarlo a cabo.

Inicialmente se encuentra un capítulo de Introducción, en el se perfilan las características y motivaciones principales de este proyecto, así como muestra una idea inicial que se irá profundizando en las secciones subsiguientes.

La sección Antecedentes ofrece una base de estudio sobre la que fundamentar el proyecto y las decisiones que serán tomadas en las secciones que le siguen. Dicha sección se centra en dos pilares fundamentales para el proyecto: la base científicomédica de la captura y análisis electromiográfico y la tecnología de hardware que lo hace posible.

Con esa base se puede continuar a la sección de Decisiones de Diseño donde se presentan diferentes alternativas para resolver el problema del sensado de la actividad muscular, el problema del control de los movimientos de la mano electromecánica y los procesos de decisión que nos llevaron a tomar uno u otro camino para su resolución.

Luego de esta, se encuentra la sección de la fase de Desarrollo del proyecto. La cual se encontrará dividida en dos secciones, una con un gran contenido técnico, centrada en la arquitectura y el diseño del dispositivo y aplicación, y la segunda, donde se ofrecerá una vista esquemática sobre el modelo del proyecto y su funcionamiento.

En la última sección se muestran los resultados obtenidos respecto a los esperados posibles en este proyecto. Presentándose también, algunas de las posibles mejoras posibles para este proyecto.

Para concluir, se dispone la bibliografía que hizo posible el desarrollo del proyecto, como también el link al código, modelo 3D y esquemáticos necesarios para reproducirlo.

2. Antecedentes

Para la realización del proyecto se ha llevado a cabo una investigación que cubre desde aspectos médicos, de manera suave, para entender el posicionamiento de los electrodos en el músculo del antebrazo hasta aspectos más técnicos como la forma en la que se iban a comunicar los datos y las plataformas utilizadas.

En esta sección se muestran los casos investigados, exponiendo un breve resumen de cada uno de ellos. De este modo se pretende dar una visión general de los aspectos básicos que ayudaron a la realización del proyecto, facilitando así el entendimiento del escrito.

2.1. Electromiografía (EMG)

La electromigráfia^[14] es la técnica de registro gráfico de la actividad eléctrica producida por los músculos esqueléticos. Siempre que un músculo en el cuerpo se contrae o se flexiona, se produce una pequeña señal eléctrica que es creada por la interacción química en el cuerpo

El EMG puede ser monitoreado a través de electrodos insertados dentro de los músculos (electrodos intramusculares) o a través de electrodos en la superficie de la piel sobre el músculo (electrodos superficiales), siendo estos últimos los elegidos para el desarrollo de este proyecto.

En su uso en medicina, el EMG es usado para estudiar el sistema neuromuscular, para el diagnóstico de enfermedades neuromusculares, y para monitorear la activación de músculos de un paciente.

2.1.1. Característica eléctrica de la EMG

La fuente eléctrica es el potencial de la membrana muscular de alrededor de -70 mV. Los rangos potenciales de EMG medidos están en el rango de entre 50 μ V hasta 20 o 30 mV, dependiendo del músculo en observación.

El potencial de membrana se refiere a la diferencia del potencial eléctrico entre el interior y el exterior de una célula y tiene un valor promedio dentro del rango de -40 mV a -80 mV.

El rango típico de repetición de una unidad motora muscular es de alrededor 7–20 Hz dependiendo del tamaño del músculo.

Una unidad motora se define como un motor neurona y todas las fibras musculares que inerva. Cuando una unidad motora se activa, el impulso llamado potencial de acción se desplaza de la neurona motora hacia el músculo. El área donde el nervio hace contacto con el músculo se llama unión neuromuscular. Después de que el potencial de acción se transmite a través de la unión neuromuscular, se obtiene un potencial en todas las fibras musculares inervadas por la unidad motora particular. La suma de toda esta actividad eléctrica se conoce como un potencial motor de la acción de la unidad (MUAP). La actividad electrofisiológica de las múltiples unidades motoras es la señal que normalmente se evalúa durante un EMG. La composición de la unidad motora, el número de fibras

musculares por unidad motora, el tipo metabólico de las fibras musculares y muchos otros factores afectan la forma de los potenciales de unidad motora en el miograma.

2.1.2. Procedimiento

Hay dos métodos para utilizar el EMG, uno es el intramuscular, y el otro método es el superficial. Para llevar a cabo un EMG intramuscular, se usa una aguja electrodo, se inserta a través de la piel hasta que entre al tejido muscular.

El método Intramuscular EMG es demasiado invasivo para el propósito de este proyecto, por lo que no se entra en detalle en su procedimiento. En su lugar, el método superficial emplea una superficie en la cual el electrodo se puede utilizar para controlar la imagen general de la activación muscular, a diferencia de la actividad de sólo unas pocas fibras como se observa utilizando un EMG intramuscular.

En el método superficial, se requieren al menos dos electrodos, debido a que lo que se obtiene como resultado es la diferencia de potencial entre cada par de electrodos. Como es de imaginar, solo sirve para el registro de músculos superficiales, debiendo tener en cuenta que las características de la piel, o el tejido adiposo debajo de la misma, pueden ocasionar problemas para medir ducha actividad muscular y no puede discriminar entre la actividad de músculos adyacentes.

Si bien este segundo método no brinda la precisión del intramuscular, resulta más cómodo al utilizarse electrodos descartables que no implican la inserción de ningún tipo de aguja a través de la piel.

Para el procesamiento de la señal EMG, es necesaria la rectificación de la señal obtenida de manera directa (raw), esto se realiza a fin de evitar un valor promedio igual a cero por los valores positivos y negativos que la señal raw puede tomar. Existen dos métodos para realizar la rectificación de la señal, el primero, Full length frequency rectification, realiza el valor absoluto de la señal, es decir, a los valores positivos se le añaden los negativos medidos, este método es el preferido ya que conserva toda la energía de la señal. El segundo método, Half length rectification, elimina del arreglo de datos los valores negativos, haciendo que el valor promedio no pueda ser cero, pero perdiendo datos de la señal.

2.1.3. Instrumental

Un equipo básico de electromiografía utilizado en medicina consta de los siguientes elementos:

- **Electrodos:** Recogen la actividad eléctrica dentro del músculo, así sea por inserción en el mismo o a través de la piel que lo cubre:

- Electrodos superficiales, son pequeños discos metálicos de material altamente conductor que se adhieren a la piel. Para reducir la impedancia entre el electrodo y la piel, se aplica un gel conductor especial. Con estos electrodos se obtiene una visión general del funcionamiento del músculo.

- Electrodos de inserción o profundos, con forma de aguja, existen varios tipos, pudiendo ser:

- Electrodos Monopulares, consisten en una aguja corriente que ha sido aislada en toda su longitud, excepto en la punta.

- Electrodos Coaxiales, consisten en una aguja en cuyo interior se han insertado conductores metálicos muy delgados, aislados entre sí y con respecto a la aguja. Sólo en la puntal los conductores no presentan aislamiento y en ese punto se captura la señal procedente del tejido muscular.

- **Amplificador:** Son necesarios para que las señales eléctricas analógicas provenientes del músculo puedan ser visualizadas en un monitor. La relación de amplificación puede superar los 60dB. El ancho de banda es de 40 a 10kHz.

En general, las características electrónicas del amplificador varían según el tipo de estudio a realizar, siendo las principales: Número de canales: 2, 4, 8. Sensibilidad: 1 pV/div. a 10 mV/div. Impedancia de entrada: $100 \text{ M}\Omega // 47 \text{ pF}$. CMRR a 50 Hz > 100 dB. Filtro pasa alto: entre 0,5 Hz y 3 kHz (6 dB/octava). Filtro pasa bajo: entre 0,1 y 15 kHz (12 dB/octava). Ruido: 1 pV eficaz entre 2 Hz y 10 kHz con la entrada cortocircuitada.

- **Sistema de registro:** Se pueden registrar las señales obtenidas del músculo en una pantalla en forma visual, y en forma sonora a través de un parlante. También se puede realizar un registro en un soporte permanente, como papel.

2.1.4. Limitaciones EMG superficial

El uso de electrodos superficiales tiene una aplicación limitada, debido a los problemas inherentes a su naturaleza. El tejido adiposo, también conocido como grasa, puede afectar las mediciones de EMG, se han realizado estudios que demuestran que a mayor presencia de grasa debajo de la piel, la actividad muscular medida es menor.

Las mediciones de EMG son más precisas en personas con un índice de masa corporal bajo, debido a la poca presencia de tejido adiposo en su organismo, así también como dependiendo de la edad, siendo menos precisa en proporción a la edad de la persona.

Como ya se ha dicho anteriormente, solo se puede detectar la actividad muscular de músculos superficiales, lo cual implica que una incorrecta colocación de los electrodos derive en una medición cruzada de actividad de dos músculos adyacentes por la dificultad de identificar a un músculo en particular.

2.2. Prótesis

Las prótesis no son algo nuevo, la historia de las mismas comenzó alrededor del año 1500 a. C. y, desde entonces, ha estado en constante evolución. Han habido muchos perfeccionamientos desde las primeras patas de palo y los primeros ganchos de mano, y el resultado ha sido la fijación y el moldeado altamente personalizados que se encuentran en los dispositivos actuales.^[20]

Los egipcios fueron los primeros pioneros de la tecnología protésica. Elaboraban sus extremidades protésicas rudimentarias con fibras, y se cree que las utilizaban por la sensación de “completitud” antes que por la función en sí. Sin embargo, recientemente, los científicos descubrieron en una momia egipcia lo que se cree que fue el primer dedo del pie protésico, que parece haber sido funcional.



Fig. 1 - Dedo prostético egipcio.

Los materiales utilizados, generalmente la madera, no sufrieron cambios importantes sino hasta el renacimiento, donde se produjo un renacer en la historia de la protésica. Durante este período, las prótesis generalmente se elaboraban con hierro, acero y cobre además de madera.

Es en el año 1508 donde las prótesis comienzan a ser más avanzadas, cuando se elaboró un par de manos de hierro para el mercenario alemán Gotz von Berlichingen después de que perdió su brazo derecho en la batalla de Landshut. Era posible manejar las manos fijándolas con la mano natural y moverlas soltando una serie de mecanismos de liberación y resortes, mientras se suspendían con correas de cuero.

Sin embargo, muchos consideran al barbero y cirujano del Ejército Francés Ambroise Paré el padre de la cirugía de amputación y del diseño protésico modernos. Introdujo modernos procedimientos de amputación (1529) en la comunidad médica y elaboró prótesis (1536) para amputados de extremidades superior e inferior. Además, inventó un dispositivo por encima de la rodilla, que consistía en una pata de palo que podía flexionarse en la rodilla y una prótesis de pie con una posición fija, un arnés ajustable, control de bloqueo de rodilla y otras características de ingeniería que se utilizan en los dispositivos actuales. Su trabajo demostraba, por primera vez, que se había comprendido verdaderamente cómo debería funcionar una prótesis. Un colega de Paré, el cerrajero

francés Lorrain, hizo una de las contribuciones más importantes en este campo cuando utilizó cuero, papel y pegamento en lugar de hierro pesado para elaborar una prótesis.

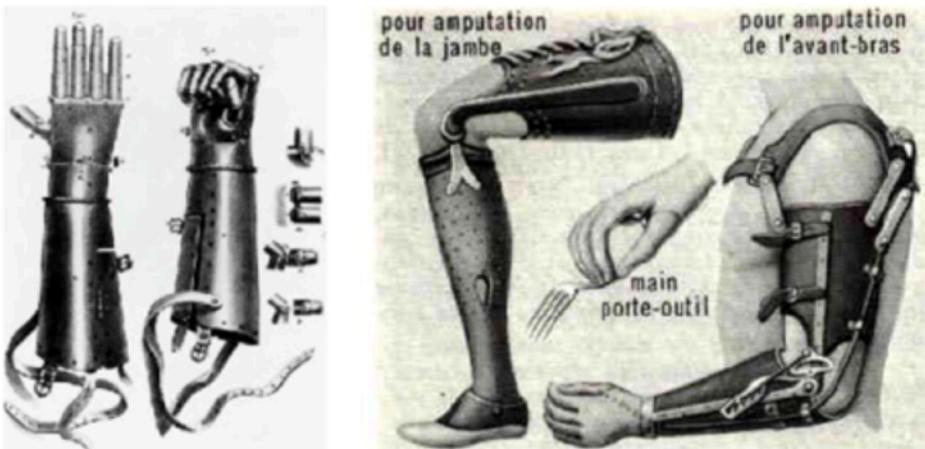


Fig. 2 - Primeras prótesis ortopédicas de Ambroise Paré.

A medida que se desarrollaba la Guerra Civil Estadounidense, la cantidad de amputados incrementaba en forma astronómica, lo que obligó a los estadounidenses a ingresar en el campo de la protésica. James Hanger, uno de los primeros amputados de la Guerra Civil, desarrolló lo que más tarde patentó como la “Extremidad Hanger”, elaborada con duelas de barril cortadas. Personas como Hanger, Selpho, Palmer y A.A. Marks ayudaron a transformar y hacer progresar el campo de la protésica con los perfeccionamientos que impusieron en los mecanismos y materiales de los dispositivos de la época.

A diferencia de la Guerra Civil, la Primera Guerra Mundial no fomentó mucho el avance en este campo. A pesar de la falta de avances tecnológicos, el Cirujano General del Ejército en ese momento comprendió la importancia del debate sobre tecnología y desarrollo de prótesis; con el tiempo, esto dio lugar a la creación de la Asociación Estadounidense de Ortoprótesis (AOPA, por sus siglas en inglés). Después de la Segunda Guerra Mundial, los veteranos estaban insatisfechos por la falta de tecnología en sus dispositivos y exigían mejoras. El gobierno de los EE. UU. cerró un trato con compañías militares para que mejoraran la función protésica en lugar de la de las armas. Este acuerdo allanó el camino para el desarrollo y la producción de las prótesis modernas. Los dispositivos actuales son mucho más livianos, se elaboran con plástico, aluminio y materiales compuestos para proporcionar a los amputados dispositivos más funcionales.

Además de ser dispositivos más livianos y estar hechos a la medida del paciente, el advenimiento de los microprocesadores, los chips informáticos y la robótica en los dispositivos actuales permitieron que los amputados recuperen el estilo de vida al que estaban acostumbrados, en lugar de simplemente proporcionarles una funcionalidad básica o un aspecto más agradable. Las prótesis son más reales con fundas de silicona y pueden imitar la función de una extremidad natural hoy más que nunca.

Al explorar la historia de la protésica, podemos apreciar todo lo que implicó la elaboración de un dispositivo y las perseverantes generaciones que hicieron falta para garantizar que el hombre pueda tener no solo las cuatro extremidades sino también la función.

2.2.1. Manos mecánicas

Gracias a la tecnología de impresión 3D dichas prótesis son de las más utilizadas debido a su bajo coste y versatilidad a la hora de construir el mecanismo de acople al usuario final.

Organizaciones sin fines de lucro, brindan dichas prótesis o bien invitan a particulares que posean una impresora 3D a realizar dichas prótesis y entregarlas a quienes lo necesiten.

Su funcionamiento se basa en la flexión de otro miembro para poder realizar, en este caso, el movimiento de abrir y cerrar la mano.



Fig. 3 - Prótesis mecánica desarrollada por AtomicLab^[18].

Por su método de construcción y material utilizado, su costo varía entre los US\$17 y US\$25, dependiendo el tipo de acoplamiento que sea necesario.

2.2.2. Manos electromecánicas

Este tipo de prótesis comienzan a ver la luz en 1946 cuando se crean sistemas de propulsión asistida. Un sistema de propulsión asistida es aquel en el que el movimiento es activado por algún agente externo al cuerpo. Las prótesis con mando mioeléctrico comienzan a surgir en el año de 1960 en Rusia. Esta opción protésica funciona con pequeños potenciales extraídos durante la contracción de las masas musculares del muñón, siendo estos conducidos y amplificados para obtener el movimiento de la misma. En sus

inicios, este tipo de prótesis solo era colocada para amputados de antebrazo, logrando una fuerza prensora de dos kilos.

Actualmente la mayoría de funciones de las prótesis de mano están limitadas al cierre y apertura de la pinza, la diferencia entre éstas radican en el tipo de control que emplean, pero todas realizan básicamente las mismas actividades. Los países con mayor avance tecnológico e investigación sobre prótesis son Alemania, Estados Unidos, Francia, Inglaterra. Una mano biónica totalmente funcional, controlada con la mente y músculos del paciente al cual se le ha sido implantada, ha sido puesta en el mercado y ha contado con una muy buena aceptación debido a su alta eficiencia al momento de interactuar con el paciente. Este nuevo diseño conocido como i-limb, fue inventado por el investigador escocés David Gow pero diseñado y fabricado por la empresa Touch Bionics. La mano funciona con un sistema de control intuitivo que recoge las señales eléctricas que generan los músculos del miembro residual del paciente. Estas señales mioeléctricas, son recogidas por electrodos que se colocan en la superficie de la piel y posteriormente son procesadas para que la mano se mueva.

Lo que hace de i-limb que sea de las más avanzadas es no solo el uso de micro motores en cada uno de los dedos para lograr un agarre lo más parecido posible al de una mano humana y estar controlada por electrodos superficiales, sino que cuenta además con una aplicación que se conecta por bluetooth para controlar los movimientos de la prótesis para casos donde los movimientos deben ser muy precisos o no son naturales.

Esta mano, siendo de lo mejor que se puede encontrar en el mercado, tiene un costo de entre U\$S 80.000 a U\$S 120.000



Fig. 4 - Prótesis i-limb ultra.

2.3. Tecnologías

Para llevar a cabo la captura de señales electromiográficas, su posterior procesamiento, graficado de las mismas y control de la mano electromecánica se han tenido que investigar diversas tecnologías.

Se ha necesitado un módulo de captura, una unidad de procesamiento, un dispositivo capaz de graficar los datos e interfaces de conexión entre ellos.

Como unidad de procesamiento se hace uso de una placa Raspberry Pi Zero W, aprovechando su bajo costo, capacidad de procesamiento y conectividad. En lo que concierne a la captura de datos, se realizó un primer análisis con el chip ADS1294^[3] de Texas Instruments que permite unas velocidades de captura elevadas y alta precisión a un coste razonable para luego utilizar el sensor MyoWare Muscle Sensor^[5] de la empresa Advancer Technologies, que se comunica a través de una interfaz i2c con la RPi. Respecto al graficado, se realizó una aplicación para iOS sobre Swift^[6], comunicada a través de BLE con la RPi corriendo un servidor GATT^[22].

2.3.1. Raspberry Pi Zero W

La Raspberry Pi^[9] es una placa de bajo coste desarrollada por la Universidad de Cambridge para promover la enseñanza de ciencias de la computación en entornos académicos. Actualmente se pueden encontrar varios modelos en el mercado. Entre ellos, los modelos Zero, que son los de tamaño más reducido.

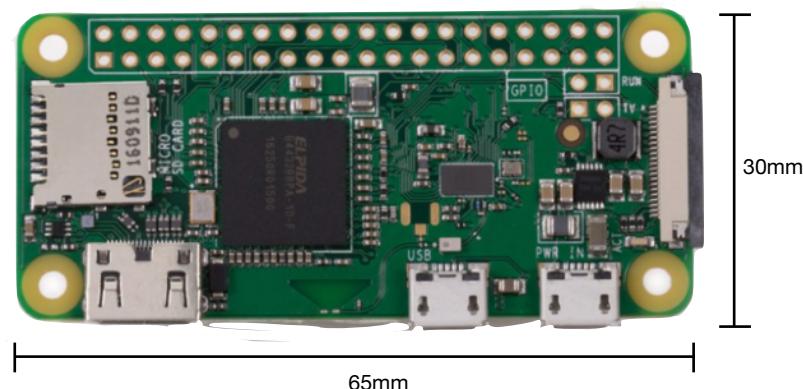


Fig. 5 - Medidas Raspberry Pi Zero W.

Los modelos Zero están constituidos por el System-on-Chip Broadcom BCM2835^[7], que utiliza como procesador un ARM11 a 1GHz (Arquitectura versión 6 de ARM), 512MB RAM y un procesador gráfico VideoCore 4. Asimismo constan de una salida mini HDMI,

un puerto USB On-The-Go, 40 pines de entrada/salida de propósito general y un conector CSI. Como dispositivo de almacenamiento de datos no volátil es necesario utilizar una tarjeta SD.

Dentro de la familia Zero, el modelo elegido, el Zero W, con un costo de aproximadamente 10 dólares, cuenta con conectividad:

- 802.11 b/g/n wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)

El consumo energético es de aproximadamente 250mA (1.25W).^[16]

La fundación Raspberrypi.org es la encargada de dar soporte a la placa, generando distribuciones de Linux, apoyando el desarrollo y fomentando el crecimiento de una comunidad en torno al dispositivo.

2.3.2. ADS1294

El chip ADS1294^[3] es un Front-End de Texas Instruments de bajo consumo para mediciones de señales biológicas como ECG, EMG Y EEG. Es un conversor analógico digital delta-sigma multicanal de sampleo simulaneo, de 24 bits, se caracteriza por tener cuatro canales con una frecuencia de muestreo de hasta 32 KHz, un amplificador de ganancia programable, referencia interna y un oscilador integrado. Cada canal consta de un multiplexor que permite la lectura desde cuatro entradas diferentes, siendo las más relevantes las entradas de temperatura, electrodos y señal de test generada internamente.

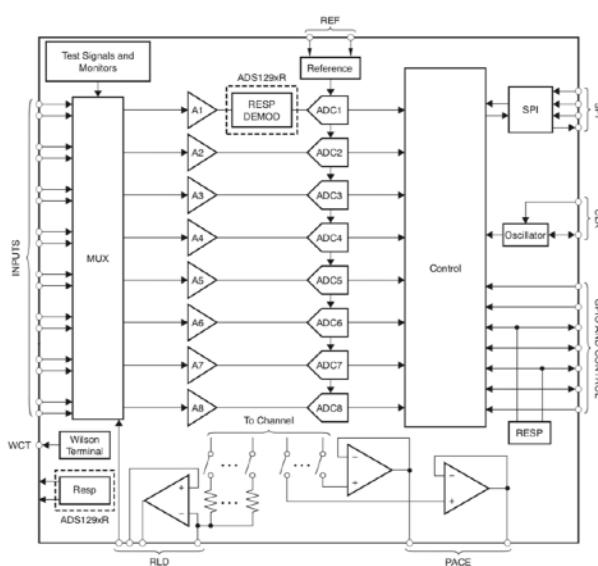


Fig. 6 - Esquemático simplificado ADS1298^[3].

El chip consta de una interfaz SPI para permitir la comunicación con otros dispositivos. Además de las líneas típicas de SPI, se proporciona una línea adicional, DRDY, que indica cuando se tienen nuevos datos válidos preparados para enviar. La lectura de datos se realiza siempre para los 4 canales, devolviendo adicionalmente una cabecera con información de la configuración del chip.

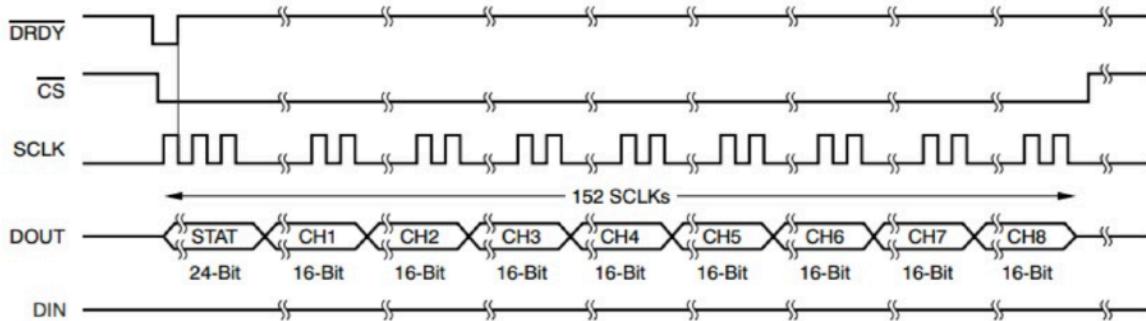


Fig. 7 - Salida del bus SPI durante lectura del chip ADS1294^[3].

El chip permite obtener la alimentación de manera unipolar o bipolar:

- Unipolar: La alimentación unipolar se realiza mediante una entrada de 5V y otra de 3V, coincidiendo estas con las proporcionadas por la Raspberry Pi mediante los pines de propósito general.
- Bipolar: El modo bipolar requiere entradas de $\pm 2.5V$. Este modo no ha sido tenido en cuenta en este proyecto por la facilidad que supone la alimentación unipolar.

Además de las entradas de SPI, el chip cuenta con 3 entradas de especial importancia para su funcionamiento:

- START: Mediante esta entrada permitimos el inicio de las conversiones. Adicionalmente podemos dejar la entrada a 0 y realizar el mismo comportamiento mediante SPI, haciendo uso de un comando específico.
- RESET: Esta entrada fuerza el estado de reset del chip. Podemos dejar la entrada a 1 y hacer uso de un comando específico mediante SPI que realiza la misma función.
- PWDN: Esta última entrada enciende o apaga el chip, por lo que si está a 0 el chip se encontrará desconectado y a 1 estará conectado y funcionando.

Internamente el ADS1294^[3] cuenta con 25 registros que permiten al usuario configurar todas las características programables del chip. Gran parte de la configuración

está relacionada con aspectos propios de la captura de señales, como las ganancias, el uso de un oscilador interno o el voltaje de referencia utilizado para la captura. El resto de configuraciones posibles permiten variar la frecuencia de captura, las entradas de los canales o modificar las señales de test internas en amplitud y frecuencia.

Dado su elevado rendimiento, alto nivel de integración y bajo consumo, el ADS1294^[3] permite el desarrollo de instrumentación médica de prestaciones elevadas, tamaño reducido y bajo coste.

Características técnicas:

- 4 canales de alta resolución y de bajo ruido.
- Bajo consumo: 0.75 mW / canal
- Frecuencia de muestreo: desde 250Hz a 32kHz.
- Ganancia programable: 1, 2, 3, 4, 6, 8, o 12.
- Alimentación: Unipolar o Bipolar.
 - Analógica: 2.7V a 5.25V
 - Digital: 1.65V a 3.6V.
- Oscilador y referencia internos.
- Señales de test integradas.
- Comunicación mediante interfaz SPI.
- Rango de temperatura operativo del encapsulado obtenido: 0°C a 70°C.

Por las razones anteriormente nombradas fue tenido en cuenta como principal opción de método de captura para el desarrollo del proyecto, en la sección Decisiones de Diseño, se enunciarán las razones por la cual el integrado descripto no fue utilizado para el desarrollo final.

2.3.3. MyoWare Muscle Sensor

Este es un sensor electromiográfico de la empresa Advancer Technologies, el cual capta la señal eléctrica generada por la pared muscular a través de tres electrodos superficiales para realizar la amplificación, rectificado e integración de dicha señal. Como se muestra en la figura 8.

En nivel de salida de la señal es de entre 0V y el nivel de alimentación, que típicamente es de 5V.

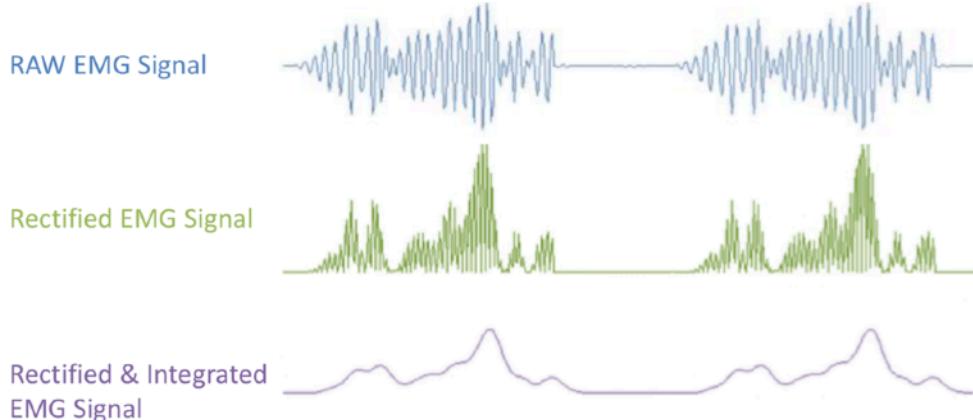


Fig. 8 - Señales intermedias y final del sensor.

Dentro de las características de este sensor se destacan su diseño Wearable, es decir, su compacto tamaño, disposición de los componentes y la posibilidad de acceder a la señal RAW a través de un pin dedicado (la cual se encuentra montada sobre una continua de Vs/2).

Posee alimentación unipolar, a diferencia de versiones anteriores del mismo, lo cual hace más sencilla su integración con otras placas y/o sensores.

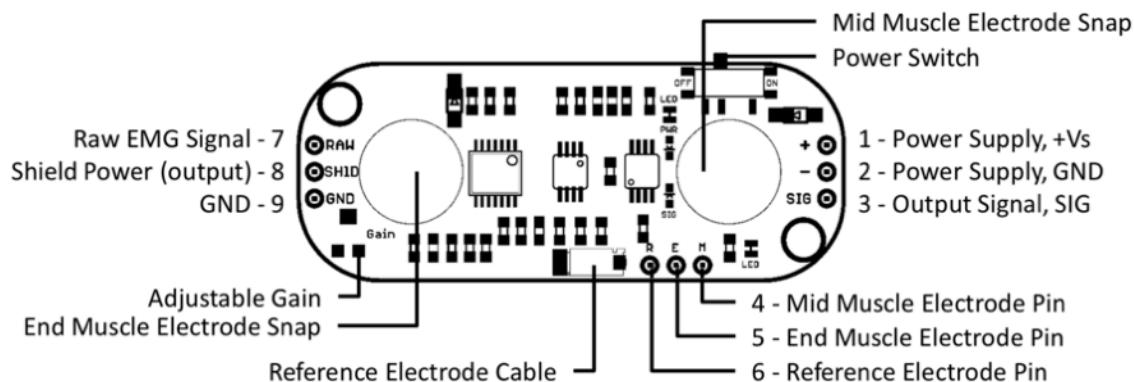


Fig. 9 - Layout Sensor muscular MyoWare AT-04-001^[5].

El AT-04-001 posee dos leds indicadores de estado, el primero se enciende para indicar que el modulo se encuentra alimentado y el segundo se enciende cuando se detecta actividad muscular significante.

Los electrodos se enganchan directamente a la placa (Como puede verse en la figura 10), lo cual hace más cómodo su manejo evitando cables adicionales y reduciendo a un mínimo errores en la colocación de dichos electrodos.

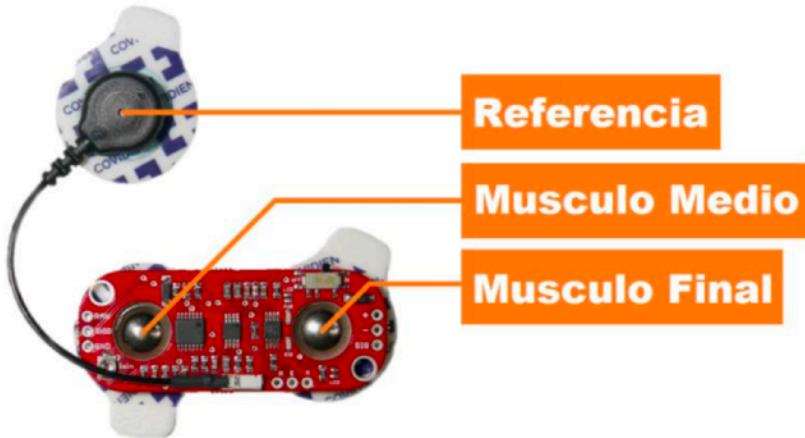


Fig. 10 - Conectores de Electrodos embebidos en la placa.

En caso de que no se desee utilizar los conectores embebidos, la placa tiene 3 pines dedicados a conectar los cables para los electrodos correspondientes.

El electrodo **medio** debe ser colocado en el medio del cuerpo del músculo.

El electrodo **final** debe colocarse adyacente al medio apuntando hacia el final del músculo.

El electrodo de **referencia** debe ser colocado en una sección diferente del cuerpo, siendo posible secciones con menos presencia muscular o en otro músculo no adyacente.

La posición y orientación de los electrodos tienen un efecto importantísimo en la fuerza y calidad de la señal medida. Estos deben ser colocados en el medio del cuerpo del músculo y alineados con la orientación de la fibra muscular, colocarlos en otra posición reduciría la intensidad de la señal y la calidad con la que es medida, debido a la reducción en el numero de unidades motoras y a la interferencia producida por los músculos adyacentes.

En la figura 11 pueden observarse las diferencias de la señal medida dependiendo de la localización del sensor sobre el cuerpo del músculo. Obtenida del manual de usuario del sensor muscular MyoWare AT-04-001^[5].

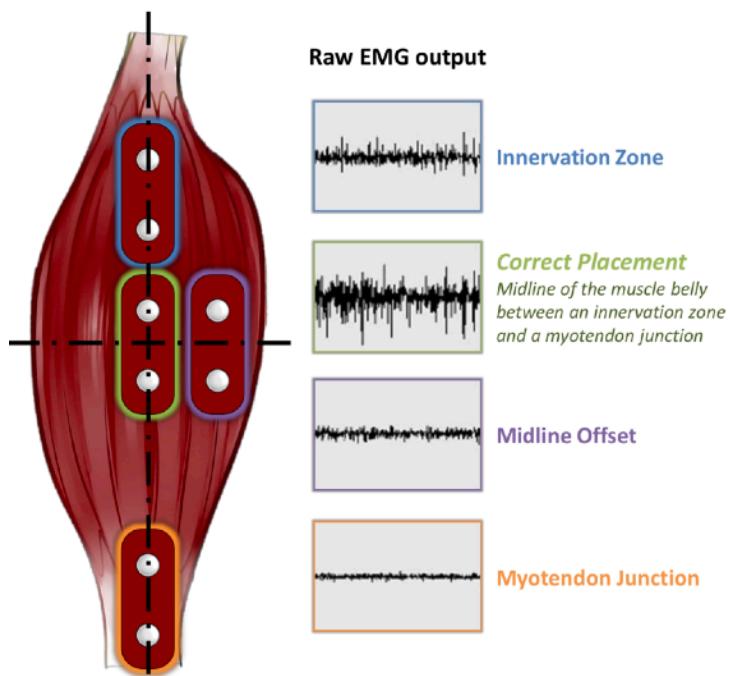


Fig. 11 - Diferentes salidas de señal dependiendo el posicionamiento del sensor sobre el músculo.

2.3.4. Bus I²C

El bus I²C (Inter-Integrated Circuit) es un bus serie de datos desarrollado en 1982 por Philips Semiconductors. Se utiliza principalmente internamente para la comunicación entre diferentes partes de un circuito, por ejemplo, entre un controlador y circuitos periféricos integrados.

El I²C está diseñado como un bus maestro-esclavo. La transferencia de datos es siempre inicializada por un maestro; el esclavo reacciona. Es posible tener varios maestros mediante un modo multimaestro, en el que se pueden comunicar dos maestros entre si, de modo que uno de ellos trabaja como esclavo. El arbitraje (control de acceso en el bus) se rige por las especificaciones, de este modo los maestros pueden ir turnándose.

El I²C precisa de dos líneas de señal: reloj (CLK, Serial Clock) y la línea de datos (SDA, Serial Data). Ambas líneas precisan resistencias de pull-up hacia VDD. Cualquier dispositivo conectado a estas líneas es de drenador o colector abierto, lo cual en combinación con las resistencias pull-up, crea un circuito Wired-AND. El nivel alto debe ser de al menos $0,7 \times VDD$ y el nivel bajo no debe ser más de $0,3 \times VDD$. Las resistencias en serie R_s , en la entrada de los dispositivos, son opcionales y se usan como resistencias de protección. El Bus I²C trabaja con lógica positiva, esto quiere decir que un nivel alto en la línea de datos corresponde a un 1 lógico, el nivel bajo a un 0.

La señal de reloj siempre es generada por el maestro. Para cada modo especificado, está predeterminado respectivamente un pulso de reloj máximo permitido. En general, también pueden ser utilizadas señales de reloj más lentas, siempre y cuando sean compatibles con la interfaz del maestro. Sin embargo, algunos circuitos integrados (por ejemplo, un conversor analógico-digital) requieren una frecuencia mínima con el fin de funcionar correctamente.

| Modo | Velocidad de transmisión máxima | Dirección |
|---------------------------|---------------------------------|----------------|
| Standard Mode (Sm) | 0,1 Mbit/s | Bidireccional |
| Fast Mode (Fm) | 0,4 Mbit/s | Bidireccional |
| Fast Mode Plus (Fm+) | 1,0 Mbit/s | Bidireccional |
| High Speed Mode (Hs-mode) | 3,4 Mbit/s | Bidireccional |
| Ultra Fast-mode (UFm) | 5,0 Mbit/s | Unidireccional |

Fig. 12 - Modos y velocidad de transmisión máxima para cada uno.

Los datos (bits individuales) sólo son válidos si su nivel lógico no cambia durante una fase de reloj alta. Las excepciones son el inicio, la parada, y la señal de inicio repetida o reset. La señal de arranque es un flanco descendente en SDA mientras SCL se encuentra en nivel alto. La señal de parada es un flanco ascendente en SDA mientras SCL está en en nivel alto. La señal de reset se comporta de igual manera que la señal de inicio.

Una unidad de datos consta de 8 bits = 1 octeto (los cuales puede ser interpretados como un valor o como una dirección, dependiendo del protocolo) y un bit de ACK. Este bit de confirmación (Acknowledge) es señalizado por un esclavo como NACK (not acknowledge) con un nivel alto, durante un nivel bajo en SDA y el noveno nivel alto de SCL (que sigue siendo generado por el maestro). El esclavo debe poner un nivel bajo en SDA antes de que SCL cambie a nivel alto, de lo contrario otros participantes podrían interpretar esto como una señal de arranque.

La dirección de I²C estándar es el primer byte enviado por el maestro, aunque los primeros 7 bits representan la dirección y el octavo bit (R/W-Bit) es el que comunica al esclavo si debe recibir datos del maestro (low/bajo) o enviar datos al maestro (high/alto). Por lo tanto, I²C utiliza un espacio de direccionamiento de 7 bits, lo cual permite hasta 112 nodos en un bus (16 de las 128 direcciones posibles están reservadas para fines especiales).

Cada uno de los circuitos integrados con capacidad de soportar un I²C tiene una dirección predeterminada por el fabricante, de la cual los últimos tres bits (subdirección)

pueden ser fijados por tres pines de control. En este caso, pueden funcionar en un I²C hasta 8 circuitos integrados. Si no es así, los circuitos integrados (que precisan ser idénticos) deben ser controlados por varios buses I²C separados.

Debido a la escasez de direcciones, se introdujo más tarde un direccionamiento de 10 bits. Es compatible con el estándar de 7 bits mediante el uso de 4 de las 16 direcciones reservadas. Ambos modos de direccionamiento pueden utilizarse simultáneamente, lo que permite hasta 1136 nodos en un único bus.

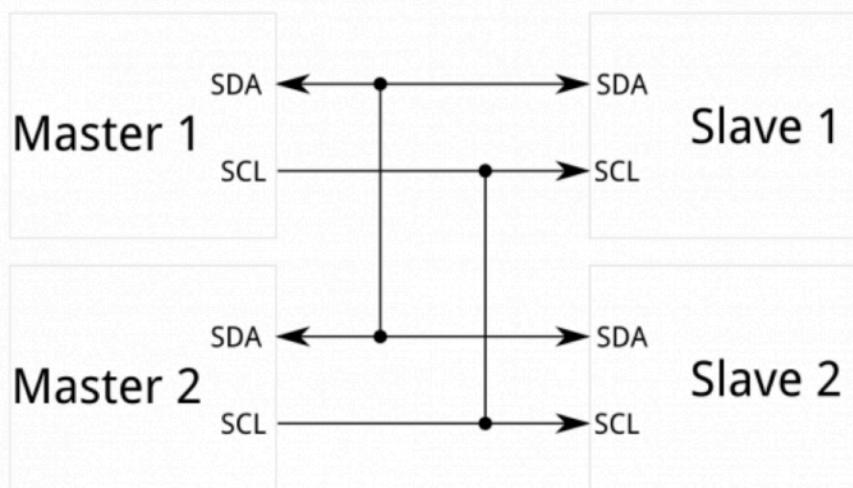


Fig. 13 - Comunicación I²C.

2.3.5. Bluetooth de baja energía (BLE)

Bluetooth Low Energy (BLE)^[21], a veces conocido como “Bluetooth Smart”, se introdujo como parte de la especificación de Bluetooth 4.0. Aunque existe cierto solapamiento con el Bluetooth clásico, BLE proviene de un proyecto inicialmente desarrollado por Nokia y conocido como ‘Wibree’ antes de que fuera adoptado por Bluetooth SIG (Special Interest Group).

Existen varios protocolos wireless para uso en IOT, pero lo que hace que BLE sea tan interesante es que es el más sencillo para implementar la comunicación entre pequeños dispositivos y una aplicación en cualquier plataforma móvil actual (iOS, Android, etc.), y particularmente en el caso de los dispositivos Apple, es el único método que permite la interacción de periféricos con aplicaciones, sin necesidad de certificaciones MFI y otros requisitos legales que exige iOS.

A continuación se realizará una pequeña introducción a BLE, sobre como se organizan los datos y como los dispositivos anuncian su presencia y la forma en la que se pueden recibir y transmitir datos.

GAP

Es el acrónimo para el Generic Access Profile, y es el encargado de controlar las conexiones y los anuncios en BLE. GAP es lo que permite que un dispositivo sea público hacia el exterior y determina como dos dispositivos pueden (o no) interactuar entre ellos.

Roles

El GAP define varios roles para los dispositivos, pero lo único que se debe tener claro es que se tendrán dispositivos centrales y los periféricos.

- Los periféricos son dispositivos pequeños, de baja potencia, de bajos recursos, que pueden conectarse a dispositivos centrales mucho más potentes. Un ejemplo de periférico puede ser un glucómetro, un medidor de pulsaciones, un beacon, entre otros.
- Un dispositivo central se corresponde normalmente con un teléfono móvil o una tablet, que tienen una capacidad de proceso mucho mayor que el periférico.

Transmisión de datos en GAP

Existen dos maneras de transmitir información a través de GAP: El Advertising Data payload y el Scan Response payload.

Ambos payloads son idénticos y pueden contener hasta 31 bytes, pero solo el Advertising Data payload es obligatorio, ya que es el payload que se transmite continuamente desde el periférico, para permitir que los nodos centrales en el alcance sepan de su presencia. El Scan Response payload es opcional y puede ser pedido desde un dispositivo central. De este modo los periféricos pueden transmitir información extra como el nombre del dispositivo o alguna característica especial definida por el fabricante.

Proceso de Advertising

Un periférico emite su Advertising Data payload a intervalos regulares configurables. Cada vez que el intervalo pasa, el periférico emite su Advertising Data payload.

Intervalos altos, permiten ahorrar batería, mientras que intervalos cortos, permiten ser más reactivos.

Si un dispositivo central necesita más datos y el periférico lo permite, puede solicitar adicionalmente el Scan Response payload, y el este contestara con la información adicional.

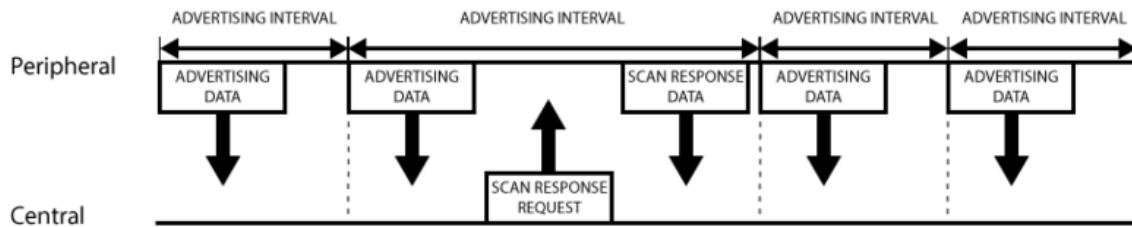


Fig. 14 - Proceso de Advertising.

El modo Broadcast

Si bien la mayoría de los periféricos se anuncian para que se pueda establecer una conexión y se puedan utilizar los servicios y las características del GATT^[22] (lo que permite intercambiar mucha más información y en ambos sentidos), hay situaciones en las que sólo se desea anunciar datos (advertising).

El principal caso de uso, es cuando se desea que un periférico transmita datos a más de un dispositivo a la vez. Esto sólo es posible utilizando el Advertising Data payload ya que los datos enviados y recibidos en modo conectado sólo pueden ser vistos por los dos dispositivos conectados.

Incluyendo una pequeña cantidad de datos personalizados en los 31 bytes del Advertising o Scan payloads, podemos usar un periférico BLE para enviar datos unidireccionalmente a dispositivos centrales en el rango de alcance. Esto es lo que se conoce como Broadcasting in BLE.

Este es el modo utilizado por la especificación iBeacon de Apple, que introduce información personalizada en el Advertising payload, usando el campo Manufacturer Specific Data.

Una vez que se establece la conexión entre un periférico y un dispositivo central, el proceso de advertising suele detenerse y usará los servicios y características del GATT^[22] para comunicarse en ambas direcciones.

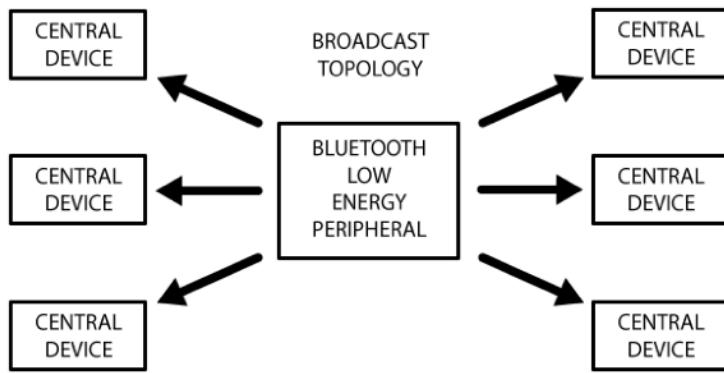


Fig. 15 - Topología del modo Broadcast.

GATT^[22]

Es el acrónimo de Generic Attribute Profile, y define la manera en que dos dispositivos BLE pueden comunicarse usando los Servicios y Características. La comunicación se realiza mediante un protocolo conocido como ATT, que se usa para almacenar los servicios, características y datos relacionados en una tabla usando identificadores de 16-bit para cada entrada en la tabla.

GATT^[22] entra en juego una vez se ha establecido una conexión dedicada entre dos dispositivos, lo que significa que ya se ha pasado previamente por el GAP.

Lo más importante a tener en cuenta con el GATT^[22] y las conexiones, es que las conexiones son exclusivas. Un periférico BLE sólo puede ser conectado a un dispositivo central a la vez. Tan pronto como un periférico se conecta a un dispositivo central, dejará de anunciarse y otros dispositivos ya no podrán verlo o conectarse a él, hasta que se finalice la conexión existente.

Establecer una conexión también es la única forma de permitir la comunicación bidireccional, en la que el dispositivo central puede enviar datos al periférico y viceversa.

El modo Connected

El siguiente figura se muestra como los dispositivos BLE funcionan en un entorno conectado. Un periférico sólo puede conectarse a un dispositivo central a la vez, pero el dispositivo central puede conectarse a varios periféricos.

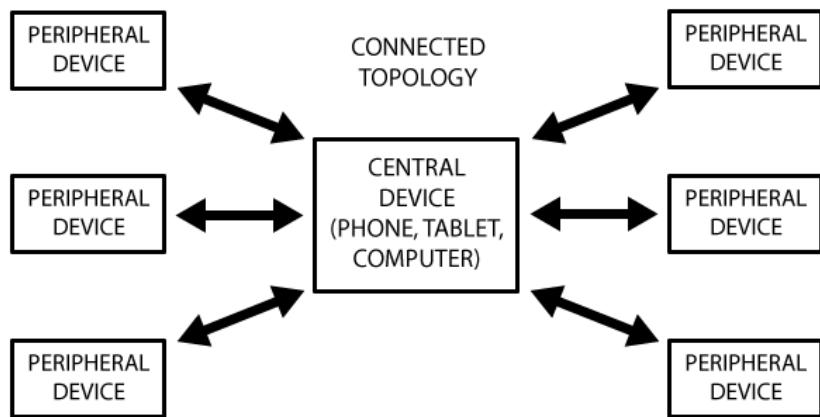


Fig. 16 - Topología del Modo conectado.

Si se requiere el intercambio de datos entre dos periféricos, sería necesario implementar un sistema de buzón personalizado donde todos los mensajes pasen a través del dispositivo central. Sin embargo, una vez que se establece una conexión entre un periférico y un dispositivo central, la comunicación puede tener lugar en ambas direcciones, lo cual es diferente al enfoque de difusión unidireccional utilizando en GAP.

Transacciones GATT^[22]

Un concepto importante para entender en GATT^[22] es la relación servidor / cliente.

El periférico se conoce como el servidor GATT^[22], que contiene los datos de búsqueda ATT y las definiciones de servicio y características, y el cliente GATT^[22] (dispositivo central), que envía solicitudes a este servidor.

Todas las transacciones son iniciadas por el dispositivo maestro, el GATT Client (central), que recibe la respuesta del dispositivo esclavo, el GATT Server (periférico).

Al establecer una conexión, el periférico sugerirá un ‘intervalo de conexión’ al dispositivo central, y el dispositivo central intentará volver a conectar cada intervalo de conexión para ver si hay nuevos datos disponibles, etc. Es importante tener en cuenta que esta conexión Intervalo es realmente sólo una sugerencia. Es posible que su dispositivo central no pueda cumplir la solicitud porque está ocupado hablando con otro periférico o los recursos del sistema necesarios no están disponibles.

El esquema del proceso de intercambio de datos entre un periférico (el Servidor GATT) y un dispositivo central (el Cliente GATT), con el dispositivo central iniciando cada transacción, se puede ver en la figura 17.

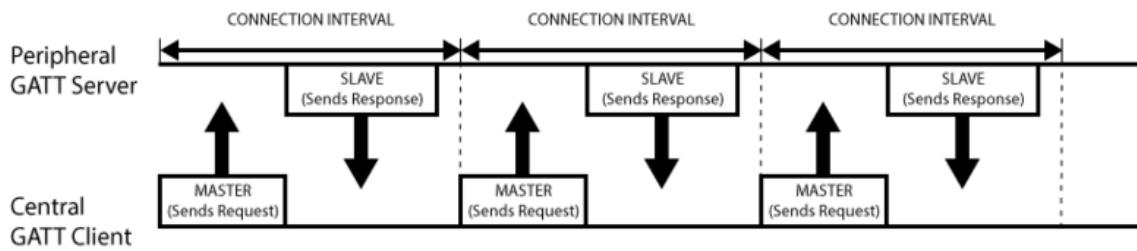


Fig. 17 - Proceso de intercambio de datos entre periféricos.

Servicios y Características

Las transacciones GATT^[22] en BLE se basan en objetos anidados de alto nivel denominados Perfiles, Servicios y Características.

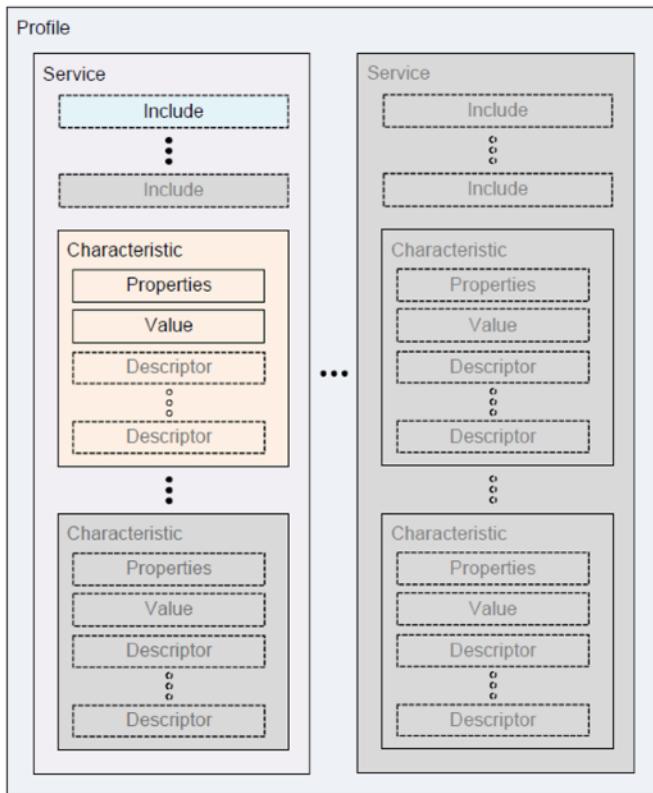


Fig. 18 - Perfiles, servicios y características de las transacciones GATT.

Perfiles

No existe un Perfil en si en el propio periférico BLE, sino que es una colección predefinida de Servicios que ha sido especificada por el Bluetooth SIG o por el fabricante del periférico.

Servicios

Los servicios se utilizan para dividir datos en entidades lógicas y contienen trozos específicos de datos llamados características. Un servicio puede tener una o más características y cada servicio se distingue de otros servicios por medio de un ID numérico único denominado UUID, que puede ser de 16 bits (para servicios BLE adoptados oficialmente) o de 128 bits (para servicios personalizados).

Se puede encontrar una lista completa de los servicios BLE adoptados oficialmente en la página del portal de desarrolladores de Bluetooth. Por ejemplo, si observamos el Servicio de Frecuencia Cardíaca, podemos ver que este servicio oficialmente adoptado tiene un UUID de 16 bits de 0x180D, y contiene hasta 3 características, aunque sólo es obligatorio la primera: medición de la frecuencia cardíaca, ubicación sensor corporal y punto de control de la frecuencia cardíaca.

Características

El concepto de nivel más bajo en las transacciones GATT son las características, que encapsulan un único tipo de dato (aunque puede contener una matriz de datos relacionados, como valores X / Y / Z de un acelerómetro de 3 ejes).

De forma similar a los Servicios, cada Característica se distingue a través de un UUID predefinido de 16 o 128 bits, y uno es libre de utilizar las características estándar definidas por el Bluetooth SIG o de definir características personalizadas que sólo el periférico creado y la aplicación pertinente entiendan.

Como ejemplo, la característica de medición de la frecuencia cardíaca es obligatoria para el servicio de frecuencia cardiaca y usa un UUID de 0x2A37. Comienza con un único valor de 8 bits que describe el formato de datos HRM (si los datos son UINT8 o UINT16, etc.), y continúa incluyendo los datos de medición de la frecuencia cardiaca que coinciden con este byte de configuración.

Las características son el elemento principal a utilizar en la interacción con el periférico BLE, por lo que es importante entender el concepto. Pueden ser de solo lectura o de escritura. De esta manera se pueden usar para realizar comunicaciones bidireccionales de una manera muy sencilla.

2.3.6. NodeJS

Node.js^[11] es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación JavaScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos, que lo hace liviano y eficiente, y basado en el motor JavaScript V8 de Google (escrito en C++). Al contrario que la mayoría del código JavaScript, no se ejecuta en un navegador, sino en el servidor.

Al ser concebido como un entorno de ejecución de JavaScript orientado a eventos asíncronos, Node está diseñado para construir aplicaciones en red escalables. Puede manejar muchas conexiones concurrentes, donde por cada conexión el callback será ejecutado, sin embargo, si no hay trabajo que realizar Node estará durmiendo.

Node.js funciona con un modelo de evaluación de un único hilo de ejecución, usando entradas y salidas asíncronas las cuales pueden ejecutarse concurrentemente en un número de hasta cientos de miles sin incurrir en costos asociados al cambio de contexto. Este diseño de compartir un único hilo de ejecución entre todas las solicitudes atiende a necesidades de aplicaciones altamente concurrentes, en el que toda operación que realice entradas y salidas debe tener una función callback. Que Node esté diseñado sin hilos, no significa que no se puede aprovechar la capacidad de los múltiples cores de un sistema actual. Procesos hijos pueden ser lanzados utilizando la API child_process.fork(), la cual está diseñada para comunicarse fácilmente con el proceso principal. Construida sobre la misma interfaz está el módulo cluster, el cual permite compartir sockets entre procesos para activar el balanceo de cargas en sus múltiples cores.

Los usuarios de Node están libres de preocupaciones sobre el bloqueo del proceso, ya que no existe. Casi ninguna función en Node realiza I/O directamente, así que el proceso nunca se bloquea. Debido a que no hay bloqueo es muy razonable desarrollar sistemas escalables en Node.

Node tiene un diseño similar y está influenciado por sistemas como Event Machine de Ruby ó Twisted de Python, siendo similar en su propósito a es similar en su propósito a libevent o libev de C. Node lleva el modelo de eventos un poco más allá, este presenta un bucle de eventos como un entorno en vez de una librería. En otros sistemas siempre existe una llamada que bloquea para iniciar el bucle de eventos. El comportamiento es típicamente definido a través de callbacks al inicio del script y al final se inicia el servidor mediante una llamada de bloqueo como EventMachine::run(). En Node no existe esta llamada. Node simplemente ingresa el bucle de eventos después de ejecutar el script de

entrada. Node sale del bucle de eventos cuando no hay más callbacks que ejecutar. Se comporta de una forma similar a JavaScript en el navegador - el bucle de eventos está oculto al usuario.

2.3.7. Swift

Swift^[6] es un lenguaje de programación multiparadigma creado por Apple enfocado en el desarrollo de aplicaciones para iOS y macOS. Fue presentado en 2014 y está diseñado para integrarse con los Frameworks Cocoa y Cocoa Touch, puede usar cualquier biblioteca programada en Objective-C y llamar a funciones de C. También es posible desarrollar código en Swift compatible con Objective-C. Swift tiene la intención de ser un lenguaje seguro, de desarrollo rápido y conciso. Usa el compilador LLVM incluido en Xcode 6 y posteriores. En el año 2015 pasó a ser de código abierto.^[23]

Es un lenguaje fuertemente tipado, aunque su declaración no siempre es necesaria gracias a su capacidad de inferir tipos. Los tipos de datos se dividen principalmente en dos grupos. Los Tipos de valor, y los Tipos por referencia, se diferencian principalmente en como son asignados:

- Al asignar un Tipo de valor se guarda una copia de su contenido. Se recomienda su uso cuando se requiere copiar su información o se vaya a trabajar en múltiples hilos.
- Al asignar un Tipo por referencia se asigna una instancia compartida que es mutable aún si son usadas en constantes, es decir modificar una instancia se verá reflejado en todas las variables y constantes que la compartan. Se recomienda su uso cuando se requiera compartir datos mutables.

Para el desarrollo de aplicaciones iOS utilizando Swift, es necesario contar con una computadora capaz de correr macOS y tener el IDE Xcode.

Xcode^[24] es un entorno de desarrollo integrado para macOS que contiene un conjunto de herramientas creadas por Apple destinadas al desarrollo de software para macOS, iOS, watchOS y tvOS. Su primera versión tiene origen en el año 2003 y actualmente su versión número 9 se encuentra disponible de manera gratuita en el Mac App Store o mediante descarga directa desde la página para desarrolladores de Apple.

Xcode trabaja conjuntamente con Interface Builder, una herencia de NeXT, una herramienta gráfica para la creación de interfaces de usuario.

Xcode incluye la colección de compiladores del proyecto GNU (GCC), y puede compilar código C, C++, Swift, Objective-C, Objective-C++, Java y AppleScript mediante una amplia gama de modelos de programación, incluyendo, pero no limitado a Cocoa, Carbon y Java.

2.3.8. Impresión 3D

La impresión 3D es un grupo de tecnologías de fabricación por adición donde un objeto tridimensional es creado mediante la superposición de capas sucesivas de material, en el caso del proyecto planteado, plástico.

Las impresoras 3D son por lo general más rápidas, más baratas y más fáciles de usar que otras tecnologías de fabricación por adición, aunque como cualquier proceso, estarán sometidas a un compromiso entre su precio de adquisición y la tolerancia en las medidas de los objetos producidos. Estas ofrecen a los desarrolladores del producto la capacidad para imprimir partes y montajes hechos de diferentes materiales con diferentes propiedades físicas y mecánicas, a menudo con un simple proceso de ensamblaje. Las tecnologías avanzadas de impresión 3D pueden incluso ofrecer modelos que pueden servir como prototipos de producto.

Desde 2003 ha crecido la comunidad de impresión 3D, haciendo que surjan nuevas y mejores tecnologías, conllevando esto a una reducción en el precio de las mismas.

Del gran número de tecnologías disponibles para la impresión 3D. Se destacan la impresión por extrusión y las de impresión por fotopolimerización, sus principales diferencias se encuentran en la forma en la que las diferentes capas son modeladas para crear cada piezas.

Por extrusión:

Usando filamentos previamente extruidos, el modelado por deposición de fundente es una tecnología desarrollada por Stratasys, usa una tobera para depositar material fundido sobre una estructura soporte, capa a capa. Es muy usada en prototipado rápido tradicional y, dado su bajo coste, se ha popularizado mucho a nivel doméstico.

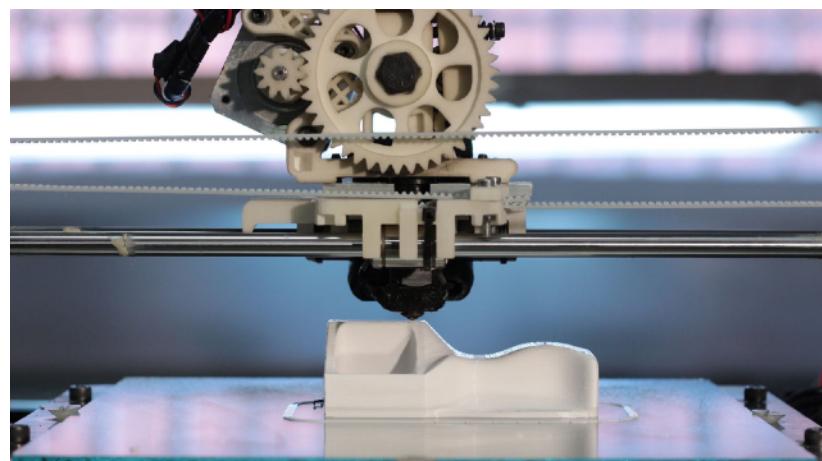


Fig. 19 - Impresión por extrusión.

Por fotopolimerización:

En esta tecnología se utilizan resinas líquidas fotopoliméricas que se solidifican cuando son expuestas a la luz emitida por un láser ultravioleta. De esta forma se van creando capas superpuestas de resina sólida que van creando el objeto.

En esta fotopolimerización por luz ultravioleta, un recipiente de polímero líquido es expuesto a la luz de un proyector DLP bajo condiciones controladas. El polímero líquido expuesto endurece; la placa de montaje se mueve hacia abajo en incrementos pequeños y el polímero es expuesto de nuevo a la luz. El proceso se repite hasta que el modelo es construido. El polímero líquido restante es entonces extraído del recipiente, dejando únicamente el modelo sólido.

Este método de impresión es bastante más rápido y posee una calidad de impresión mucho mejor que la impresión por extrusión.

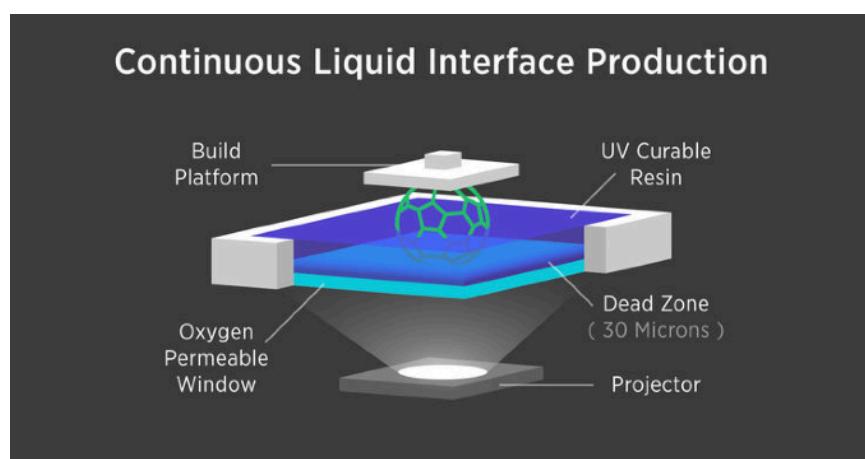


Fig. 20 - Impresión por extrusión.

3. Criterios de diseño

En esta sección se abarcarán todas las decisiones que se han tomado a la hora de realizar el diseño del sistema. Se justificarán decisiones que se han tomado a lo largo del desarrollo del proyecto, abarcando aspectos técnicos y funcionales.

Fueron investigados los diferentes sistemas operativos posibles a utilizar sobre la RPI, el flujo e interacción del sistema y las librerías posibles para realizar los gráficos correspondientes a la EMG en la aplicación.

A su vez se presentan los modelos 3D estudiados y un bosquejo del sistema que realiza la electromiografía, el cual controla la mano prostética.

3.1. Raspbian

El sistema operativo elegido a utilizar es Raspbian, siendo este una distribución de Linux que ofrece la propia RaspberryPi Foundation. Esta distribución es de fácil instalación, gratuita y consta de una comunidad activa de usuarios, encajando de este modo perfectamente con los ideales del proyecto.

Raspbian está basado en Debian Wheezy y ha sido optimizado para la Raspberry Pi. Consta de más de 35.000 paquetes, software pre-compilado y su desarrollo sigue activo.

Destacan su alta estabilidad y elevado rendimiento, habiéndose optimizado el cálculo en coma flotante mediante hardware.

3.2. Sensor muscular

Originalmente se planteó la utilización del integrado ADS1294^[3] de Texas Instruments, debido a sus características, mencionadas en la segunda sección de este informe, pero debido a complicaciones que surgieron durante el desarrollo se buscaron otras opciones de conversores analógicos digitales aptos para este tipo de requerimientos, como el ADS1294^[3] también realizaba las funciones de amplificación y filtrado, debía cumplirse que el reemplazo del integrado también realizara dichas funciones. Encontrándose así el sensor electromiográfico MyoWare^[5], que es la tercera versión de sensores de este tipo que realiza la empresa Advancer Technologies, como este sensor no dispone de un conversor AD, se coloca en serie un Atmega328^[4] que cumple la función faltante y se comunica con la RPI a través de i2c.

Al ser open-hardware, la empresa brinda de manera gratuita los esquemáticos, PCB y lista de componentes de la placa. La decisión que se tomó, por cuestiones de practicidad, teniendo en cuenta la baja disponibilidad de componentes smd en la ciudad, o el costo que tienen dichos componentes en buenos aires, o en el exterior tras sumarle envío e impuestos, se decidió comprar la placa final directamente a un distribuidor autorizado de Advancer Technologies.

Otro factor que influyó a esta decisión, fue la experiencia vivida durante la realización de la primera placa para el ADS1294^[3], donde tuvieron que pedirse todos los componentes, y realizar el desarrollo de la placa, en el exterior. Gracias a los costos de envío e impuestos, el desarrollo perdía su enfoque original de mantener el costo en lo menor posible.

Acoplados al sensor, se utilizarán electrodos superficiales, para que el sistema no sea invasivo.

3.3. Programación

3.3.1. Sobre la Raspberry

Sobre la RPI, el lenguaje de programación elegido fue JavaScript, para su utilización sobre NodeJS^[11], debido a la disponibilidad de diferentes módulos posibles para utilizar en el desarrollo de un servidor GATT, requerido para la comunicación entre la Raspberry, encargada del control de la mano electromecánica y la captura de datos del sensor, y la aplicación, donde se visualizarían los datos o se realizaría la configuración del movimiento a realizar por la aplicación.

El servidor GATT se realiza utilizando el modulo BLENO^[12], otros modelos utilizados fueron el i2c^[13], para la comunicación a través del protocolo con el mismo nombre, entre el conversor analógico digital y la RPI, y el pigpio, para poder utilizar el GPIO de la placa en cuestión.

Para correr el programa realizado, es necesario realizar un pequeño script .sh que se ejecute cada vez que se inicia el sistema.

3.3.2. Aplicación móvil

El lenguaje de programación para realizar la aplicación es Swift, debido a que se trabajará de manera nativa sobre un dispositivo iOS.

De la aplicación se espera que tenga una interfaz sencilla y que requiera la menor configuración posible.

Se parte de la base de que en esta se mostrarán los datos medidos por el sensor muscular, en forma de gráfico y se ofrecerán opciones de configuración de gestos en la misma pantalla, sin la necesidad de entrar en ningún tipo de menús.

Los frameworks principales a utilizar son CoreBluetooth, desarrollado por apple, para la comunicación BLE y Charts, para el graficado de los datos sensados.

3.4. Modelo 3D

Con respecto al modelo de mano prostética a utilizar, se analizaron varias posibilidades, de las cuales 3 fueron las tenidas en cuenta en un análisis final, estas están disponibles de manera gratuita en el sitio <https://www.thingiverse.com>, una plataforma donde los usuarios cargan los modelos 3D de manera gratuita, con la posibilidad de recibir una donación o mención.

La primera opción, figura 21, si bien presenta una palma de tamaño reducido y liviana, requiere de la sección del antebrazo para alojar la electrónica y serios que realizan la apertura /cierre de la mano. Debido a esto fue descartada.



Fig. 21 - Robotic Prosthetic Hand by grossrc.

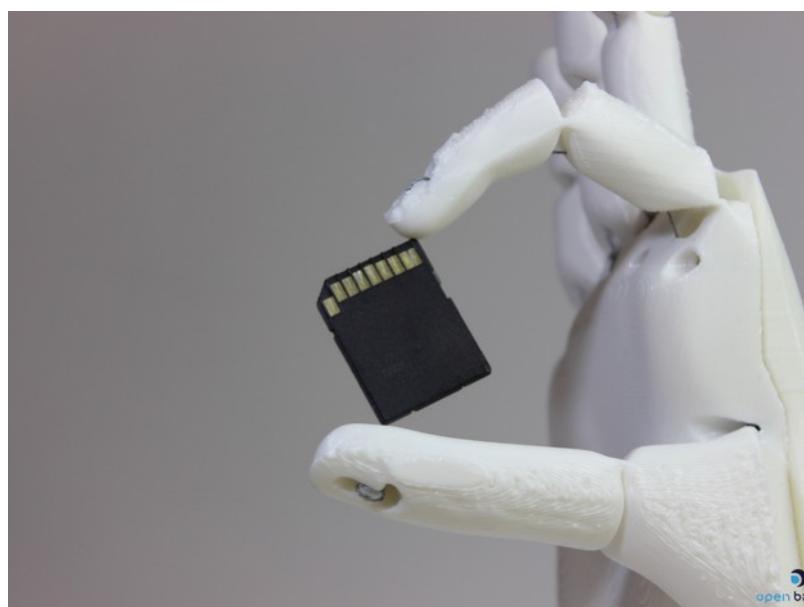


Fig. 22 - Ada Robotic Hand by openbionics.

El diseño de la figura 22, de la empresa openbionics^[26], es mucho mejor que el primero, siendo su característica principal la de utilizar material flexible para la impresión de los dedos y junturas, permitiendo que el modelo conste de un mayor agarre y flexibilidad. Dicho material flexible presenta dificultades en su impresión si no se tiene una impresora capaz de imprimirla correctamente. Esto y la utilización de actuadores lineales Firgelli de 12V, los cuales son difíciles de conseguir, hizo que esta opción también sea descartada.

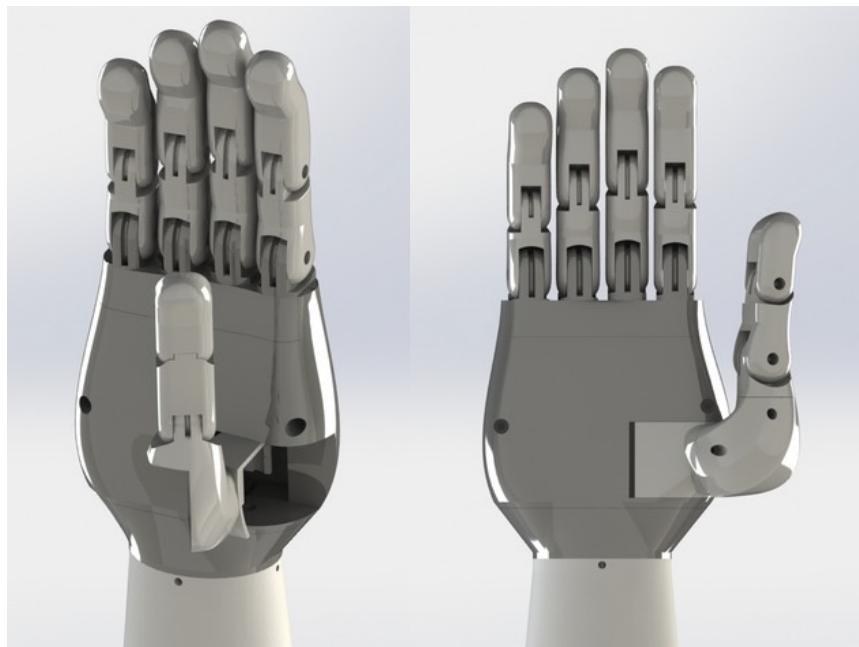


Fig. 23 - MyBio Hand v1.3 by LuisMurillo^[8].

La tercera opción analizada, la mostrada en la figura 23, que resultó la elegida, presenta una facilidad de armado y de impresión sin la utilización de materiales flexibles y para su movimiento utiliza servomotores de 9g los cuales son económicos y fáciles de conseguir.

A modo informativo se incluye también un cuarto modelo, mostrado en la figura 24, que si bien fue investigado, se decidió no tomarlo como caso a seguir, debido a la complejidad por la cantidad de partes necesarias para su desarrollo, es el modelo Exiii de hackberry^[27], el cual utiliza tanto actuadores lineales como servomotores para realizar los diferentes movimientos de la mano.

Como puede verse en la figura, los detalles de la misma hacen que luzca mucho mejor que los otros modelos mencionados, pero también requerían de materiales extras como elásticos y resortes, con las implicancias que ello conlleva, esta diseñada también

para colocar diversos botones en el dorso de la mano, para controlar los movimientos que la misma realizará al ser activada por los sensores musculares.



Fig. 24 - Exiii by hackberry.

3.5. Modelo del sistema EMG

Los electrodos serán colocados sobre el antebrazo del usuario (aunque es posible aplicarlos sobre cualquier cuerpo muscular), el AT-04-001 se encargará de la amplificación y filtrado de la señal EMG producida por dichas unidades motoras.

La señal de salida del sensor será convertida a digital a través del ADC de 8 bits presente en el integrado Atmega328p^[4], que se comunicará a su vez con la RPI a través del protocolo I²C.

El valor recibido por la RPI será utilizado para decidir si deben actuar o no los servomotores y para actualizar la característica del servicio sobre el servidor GATT que se encuentra corriendo en ella, para ser leída por la aplicación iOS corriendo sobre el dispositivo móvil y así poder realizar el gráfico de los datos.

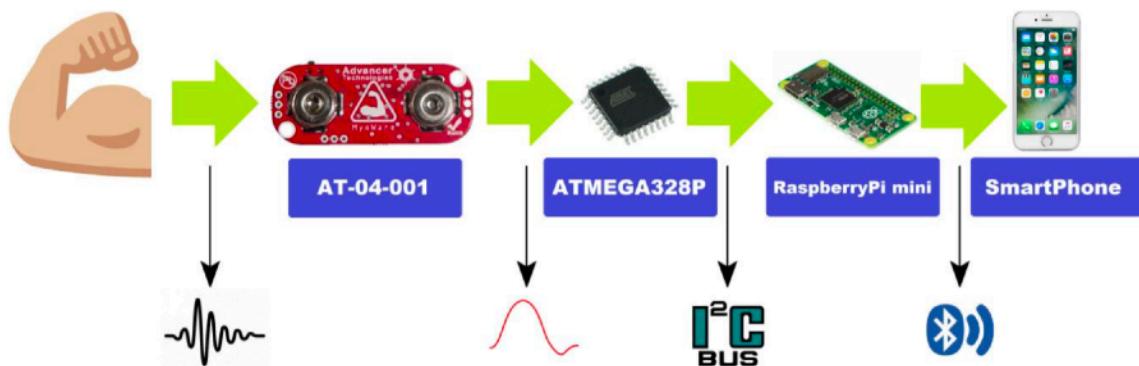


Fig. 25 - Diagrama del sistema EMG.

3.6. Conclusiones de diseño

Una vez realizada la labor de investigación previa, se plantea un proyecto que cumpla los siguientes objetivos:

Calidad de captura y análisis: La captura esperada será de un 100Hz, permitiendo que la señal sea correctamente maestreada permitiendo analizar de manera correcta la actividad muscular.

Facilidad de adquisición y uso: Haciendo que sea fácil reproducir el proyecto, cualquier persona podría desarrollar el suyo propio.

Bajo coste: Los componentes utilizados en el proyecto no tendrán un coste excesivo, la mano como prótesis es utilizable sin la necesidad de estar conectada a un dispositivo móvil. El objetivo principal del proyecto es que sea tan asequible como sea posible.

El diagrama pensado para el dispositivo implica lo siguiente:

Possible añadido de sensores musculares, si se deseara, se podrían agregar más sensores en diferentes cuerpos musculares, y el software corriendo sobre la Raspberry podría adaptarse para tomar los datos de los mismos a través de I²C y utilizarlos para realizar diferentes movimientos adicionales.

Almacenamiento permanente de datos, todas las capturas realizadas podrán almacenarse tanto como una imagen de la gráfica como en una planilla de datos adaptando el código de la aplicación móvil.

Interacción sencilla con el usuario, la experiencia de uso debe mantenerse lo suficientemente agradable y sencilla como para que su uso no suponga una molestia y no sea difícil entender el funcionamiento del sistema.

4. Desarrollo

En esta sección se explica como se han implementado los diferentes componentes del sistema y su funcionalidad. Se expone como esta desarrollado el código en base a la investigación llevada a cabo, el proceso y la planificación que se ha seguido.

4.1. Estructura del sistema

La estructura del sistema está formada por cuatro secciones independientes, que se subdividirán para dar una mejor explicación:

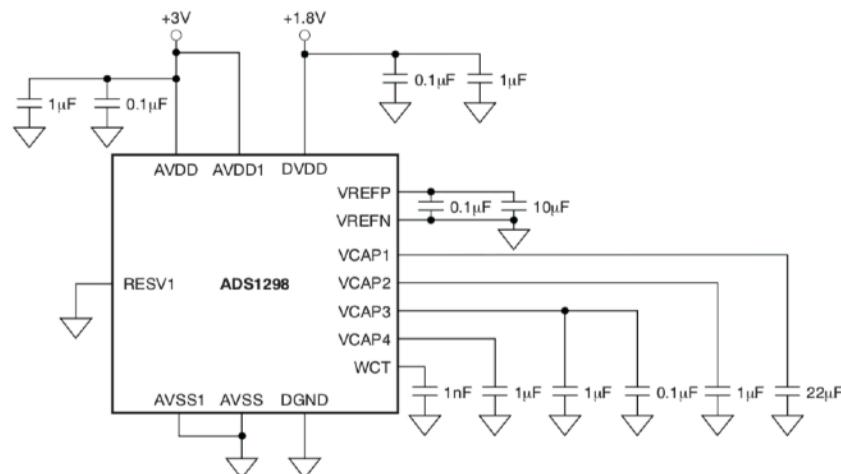
- Medición de la actividad muscular y comunicación mediante I²C.
- Procesamiento de datos y comunicación mediante BLE.
- Visualización de los datos.
- Movimiento de la mano prostética.

4.1.1. Medición de la actividad muscular

Etapa inicial - ADS1294

Como se explicó anteriormente, se comenzó a trabajar con el integrado nombrado, con el objetivo de lograr una especie de HAT^[10] para la Raspberry Pi, un HAT^[10] es una placa add-on, que se realiza siguiendo una serie de reglas de diseño propuestas por la Raspberry foundation, para las RPI desde la B+ en adelante, con la idea que se coloque sobre la pinera GPIO de la Raspberrys. Como en este caso se utiliza una RPI Zero, el tamaño de la misma debía ser de a lo sumo de 65x30mm.

Para la alimentación del integrado se siguió el esquema propuesto por Texas Instruments para alimentación unipolar (figura 26), se eligió este tipo de alimentación debido a la facilidad que suponía contar con la alimentación que proporciona la RPI.



NOTE: Place the capacitors for supply, reference, WCT, and VCAP1 to VCAP4 as close to the package as possible.

Fig. 26 - Operación unipolar - Datasheet ADS129X^[3].

De la Raspberry se pueden obtener 5V y 3.3V, como se puede observar en la figura 27. Dado que los valores extremos de la diferencia entre AVDD y AVSS van desde los -0.3V a los 5.5V y los de DVDD a DGND parten de -0.3V a 3.9V, tomando AVSS Y DGND como GND, 5V y 3.3V se corresponden correctamente con los valores posibles para AVDD Y DVDD respectivamente.

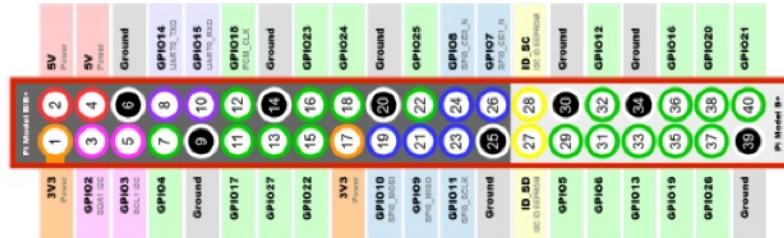


Fig. 27 - Pinout Raspberry Pi Zero W.

Los 5V entregados por los pines 2 y 4 de la RPI están conectados directamente a la alimentación, sin ningún otro tipo de filtrado adicional que el de la fuente de alimentación, para evitar usar el regulador de la placa a 3.3V se decidió que partiendo de los 5V entregados por la fuente, generar una propia alimentación regulada de 3.3V para alimentar DVDD.

Una vez resuelto lo de la alimentación, se procedió utilizando el ejemplo de layout proporcionado por Texas Instruments (Mostrado en la figura 28) para la elección y configuración de los demás componentes necesarios para la placa electromiográfica.

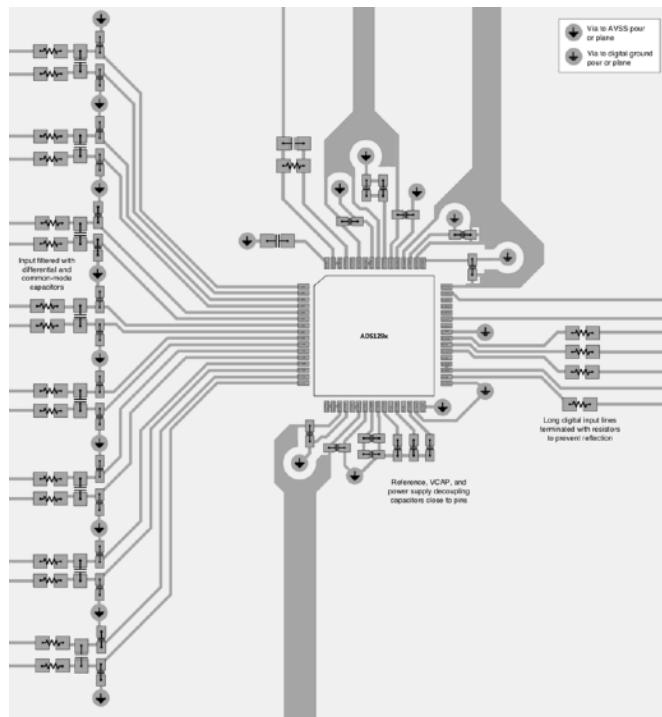


Fig. 28 - Ejemplo de layout para electrocardiograma utilizando ADS129X^[3] - Texas Instruments.

A pesar de que el layo es para una placa electrocardiográfica se realizó el esquemático mostrado en la figura 29.

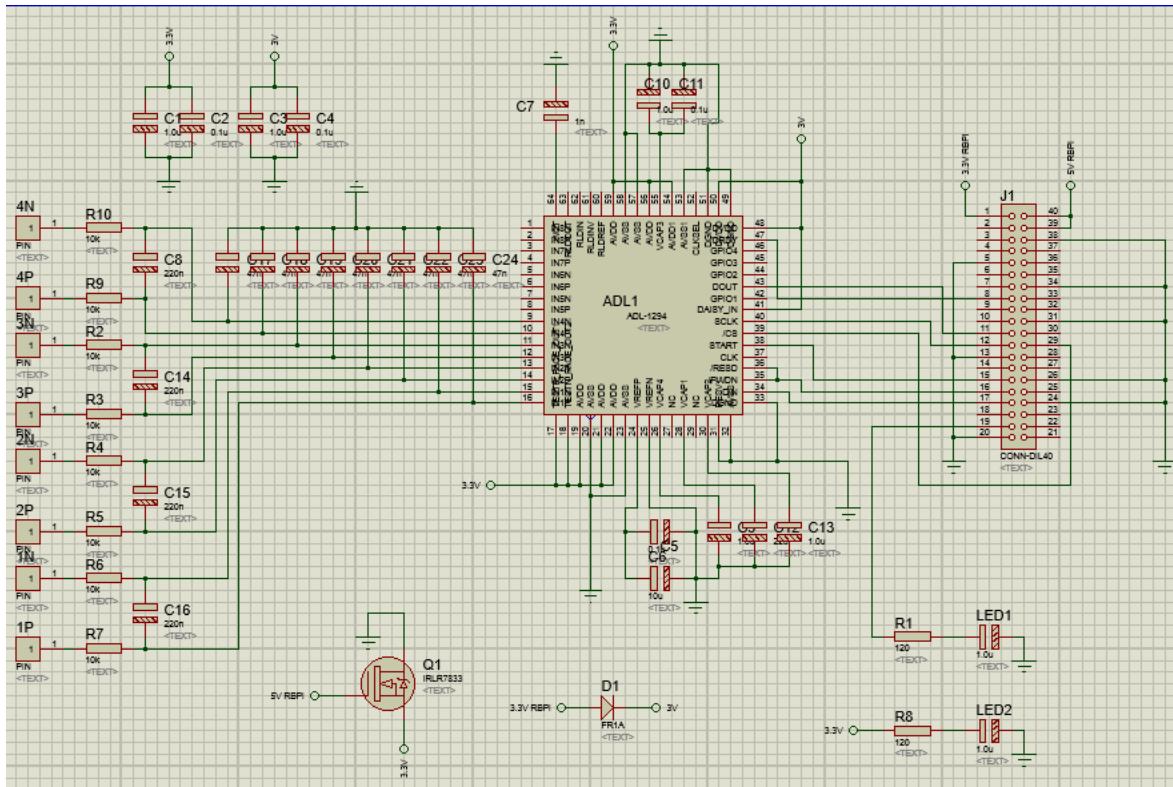


Fig. 29 - Esquemático HAT ADS1294.

Donde además de los componentes necesarios para alimentación, generación de la alimentación interna de 3.3V, comunicación SPI y funcionamiento, se le agregaron dos leds de estado, para indicar que la la placa estaba recibiendo alimentación y que el programa corriendo sobre la RPI estuviera funcionando.

Una vez localizados los componentes, en una placa de 65x30mm doble faz, como se muestran en el pcb de la figura 30:

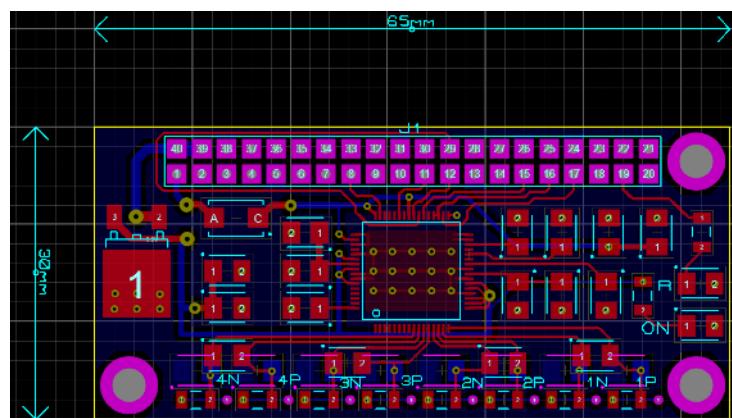


Fig. 30 - PCB HAT ADS1294.

Se realizó el pedido de cotización por la realización de la cantidad mínima de unidades posibles en dos empresas de Buenos Aires, como los costos eran extremadamente elevados, se decidió enviar la placa para su fabricación a china.

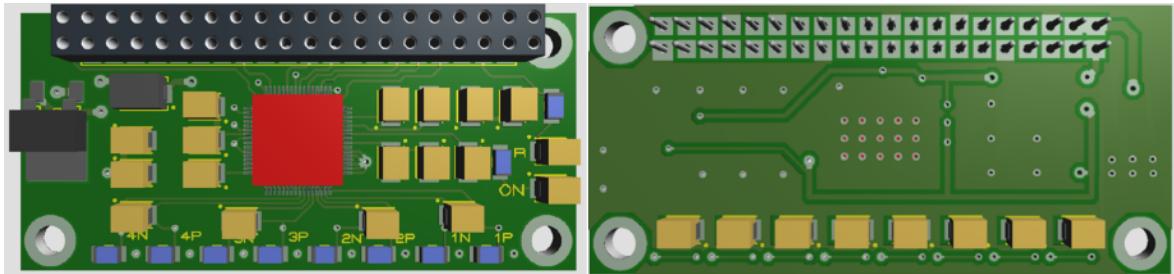


Fig. 31 - Expectativa de la placa diseñada Top / Bottom.

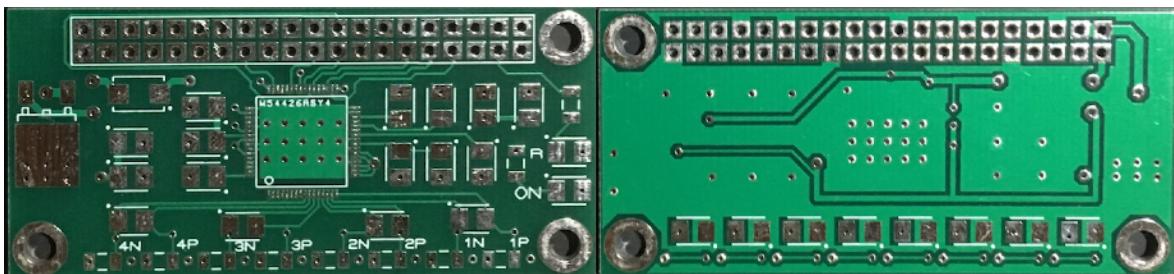


Fig. 32 - Placa recibida Top / Bottom.

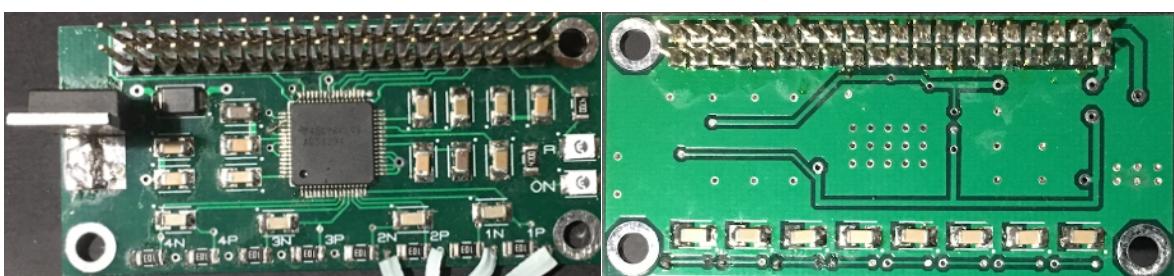


Fig. 33 - Placa soldada Top / Bottom.

El proceso de fabricación y transporte demoró tan solo 10 días, pero se demoraron 7 días más en su obtención debido a trámites aduaneros. Luego del pago de envío, impuestos y tasas, las 10 placas, resultaron costando menos de la mitad que lo presupuestado por 5 en Buenos Aires.

Luego de los primeros tests con la placa realizada, los resultados no fueron satisfactorios, si bien ambos leds de estado se encontraban encendidos, no se logró la comunicación con el integrado. Luego de varias semanas de pruebas sin éxitos se optó por investigar otra alternativa como el sensor electromiográfico MyoWare^[5].

Sensor MyoWare

Este sensor de Advancer Technologies, cuenta con la característica de ser Open Hardware, por lo que los esquemáticos pueden encontrarse libremente en internet para el desarrollo de la placa.

Al tener la posibilidad de comprar la placa final, con los conectores para los electrodos ya integrados, se optó por realizar la compra en lugar de incurrir nuevamente en el desarrollo y envío a fabricar de una placa.

Como la alimentación típica es de 5V, se realizó la conexión a la alimentación de la Raspberry Pi a fin de utilizar los mismos 5V.

Este sensor no puede ser conectado directamente a la RPI, debido a que la señal de salida es una señal analógica y la Raspberry no posee conversores analógico digitales en su GPIO, por lo que se debió conectar a un Atmega328^[4] a fin de utilizar un conversor asequible para realizar la conexión a través de I²C.

El sensor fue colocado en el músculo flexor cubital del carpo, ubicado en el antebrazo. Este se debe colocar lo más centrado al músculo posible, para obtener una lectura precisa.

4.1.2. Comunicación entre dispositivos I²C

El paso de datos y las etapas de ejecución están definidas por el programa corriendo sobre la RPI, que se encarga de realizar el pedido de los datos cada un tiempo determinado (10ms).

Siendo la Raspberry el maestro de la comunicación por I²C entre ella y el Atmega328^[4] encargado de la conversión AD, es la que le pide los datos correspondiendo con el timer que maneja internamente el código.

El integrado de atmel, tiene la dirección I²C configurada 0x04, el código que ejecuta se encarga de realizar la conversión de la entrada analógica (señal de salida del sensor EMG) a un valor digital de 8 bits, que será el enviado a la RPI mediante el protocolo mencionado.

En la raspberry, luego de la instalación de los drivers y realizar la configuración correspondiente, el código corriendo sobre NodeJS^[11] declara la comunicación I²C^[13] de la siguiente manera:

```
var address = 0x04;  
var wire = new i2c(address, {device: '/dev/i2c-1'});
```

Y para realizar la lectura del valor, se debe utilizar:

```
wire.read(1, function(err, res) { ... });
```

Donde el 1 representa la longitud del buffer en bytes.

4.1.3. Procesamiento de datos

Debido a que el filtrado de la señal se realiza en el señor muscular, dentro de la Raspberry se trabaja ya con el valor leído convertido a digital.

Dentro del programa, el valor leído del sensor, es transformado a decimal por NodeJS^[11]. Con dicho valor se evalúa si debe, o no, realizarse un movimiento en la mano prostética, cuando la decisión tomada es positiva, se analiza cual de los movimientos posibles es el que se debe realizar.

Además de utilizarse el control de la mano, dicho valor es utilizado para actualizar el valor de la característica utilizada por el servicio en el servidor GATT.

4.1.4. Comunicación entre dispositivos Bluetooth Low Energy

La comunicación BLE entre la Raspberry Pi y la aplicación móvil requiere unos pasos más para su configuración que la comunicación I²C antes explicada.

Para poder crear un servidor GATT sobre NodeJS^[11] en la RPI, se ha utilizado la librería BLENO^[12], la cual posee las funciones necesarias para hacer esto posible.

```
var BlenoPrimaryService = bleno.PrimaryService;  
var EchoCharacteristic = require('./characteristics');
```

Se crean el servicio y la característica correspondiente.

Luego se define el evento “on” del servidor bluetooth, donde se declara el servicio de echo con el uuid correspondiente, en este caso ec00.

```

bleno.on('stateChange', function(state) {
    if (state === 'poweredOn') {
        bleno.startAdvertising('echo', ['ec00']);
    } else {
        bleno.stopAdvertising();
    }
});

```

Una vez realizada la conexión entre la RPI y el teléfono celular, entra en juego la característica del servicio del GATT que se asocia al servicio principal, declarada de la siguiente manera:

```

var BlenoCharacteristic = bleno.Characteristic;
util.inherits(EchoCharacteristic, BlenoCharacteristic);

```

Y luego el uuid declarado pasa a ser ec0e, de lectura, escritura y notificación, esta última de gran utilidad para la comunicación realizada.

```

var EchoCharacteristic = function() {
    EchoCharacteristic.super_.call(this, {
        uuid: 'ec0e',
        properties: ['read', 'write', 'notify'],
        value: null
    });

    this._value = new Buffer(0);
    this._updateValueCallback = null;
};

```

Esta característica, sobre BLENO^[12], pose de cuatro funciones inherentes:

```

EchoCharacteristic.prototype.onReadRequest()

```

Utilizada para el caso de que se desee leer el valor de la característica de manera manual.

```

EchoCharacteristic.prototype.onWriteRequest()

```

Que es la que responde cuando el servidor GATT recibe datos provenientes de la aplicación.

Dentro del callback de esta función se realiza la configuración de los diferentes movimientos de la mano o el threshold mínimo para realizar el movimiento, dependiendo del valor seteado en la aplicación.

Luego, las ultimas dos funciones, las de suscripción a la notificación del cambio del valor de la característica por parte del dispositivo móvil:

```
EchoCharacteristic.prototype.onSubscribe()
EchoCharacteristic.prototype.onUnsubscribe()
```

Son las encargadas de lanzar los timers (o deshabilitarlos, respectivamente) correspondientes a la lectura de datos del sensor y a la actualización del valor de la característica de este servicio.

Dichas funciones pueden ser solo activadas si el cliente GATT se suscribe al valor de la característica, de lo contrario no serán ejecutadas por el servidor.

Del lado de la aplicación iOS, para poder funcionar como cliente GATT, se necesitan declarar las variables:

```
var manager : CBCentralManager!
var miBT : CBPeripheral!
var miCharacteristic : CBCharacteristic!
```

Definidas por el framework CoreBluetooth.

Luego de realizar el escaneo de dispositivos y encontrar el indicado, se establece la conexión, para luego escanear los servicios del mismo y las características de dicho servicio. Una vez que se cumplen las condiciones de que sean los valores esperados y se clique en el botón On de la aplicación se realiza la suscripción a la característica deseada a través de la función: setNotifyValue(true, for: miCharacteristic), para desuscribirse (al cliquear el botón off) se hace el llamado a la misma función, pero con el primer parámetro en false setNotifyValue(false, for: miCharacteristic).

4.1.5. Aplicación iOS

La aplicación fue diseñada para que la interfaz sea lo más sencilla posible y se muestre solo lo relevante para el usuario. Puede verse la pantalla principal de la aplicación en la figura 35.

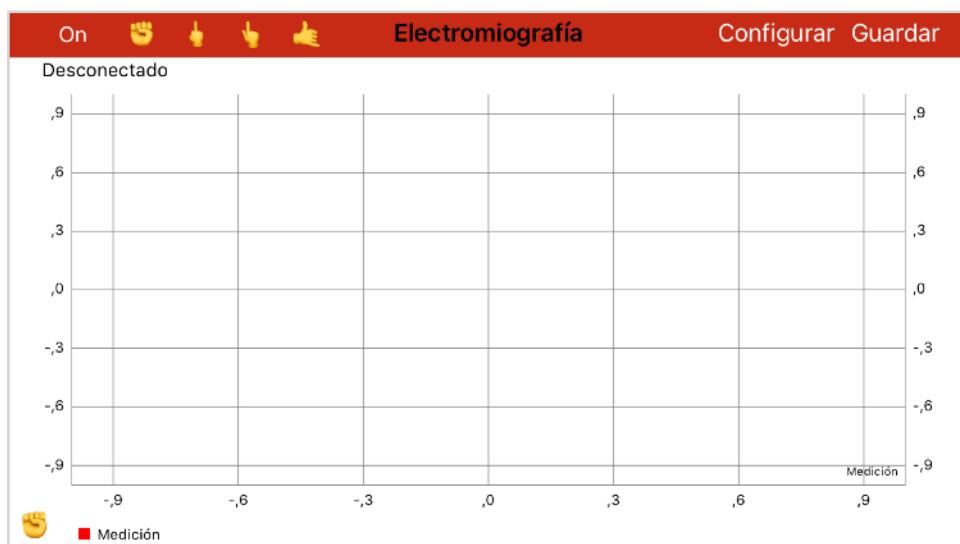


Fig. 34 - Vista principal aplicación iOS.

Se puede observar que la pantalla esta orientada al gráfico de la señal medida por el sensor muscular, agregándose los botones pertinentes a los gestos que se deseé realizar con la mano.

El botón de on, es utilizado para comenzar con el intercambio de datos con el servidor GATT, siempre y cuando el estatus sea conectado, este cambiará a off para indicar que al hacerle click finaliza la comunicación.

Los botones de gestos, enviarán la configuración necesaria a la RPI para el movimiento de la mano y cambiarán el indicador de gesto seleccionado en la esquina inferior izquierda de la pantalla.

El botón de configuración permite cambiar el nivel de threshold con el cual el sistema toma la decisión de si se debe realizar un movimiento de la mano prostética.

La aplicación también cuenta con el botón de guardar el cual guarda en el dispositivo una imagen de la curva que se visualiza en pantalla.

El framework utilizado para el graficado de los datos es Charts, que es de las más utilizadas para esta funcionalidad, luego de esta, todos los frameworks utilizados son los proporcionados por Apple.

4.1.6. Construcción de la mano prostética

Para la realización de la mano prostética se utilizó la tecnología de impresión 3D como se comentó anteriormente, utilizando un modelo 3D de licencia libre. Se utilizó PLA como material de impresión, por las cualidades mecánicas respecto al ABS (el cual es más rígido, por lo que los golpes lo soporta mejor el PLA), otra ventaja es que el PLA no produce gases tóxicos durante la impresión y es un material de procedencia vegetal.

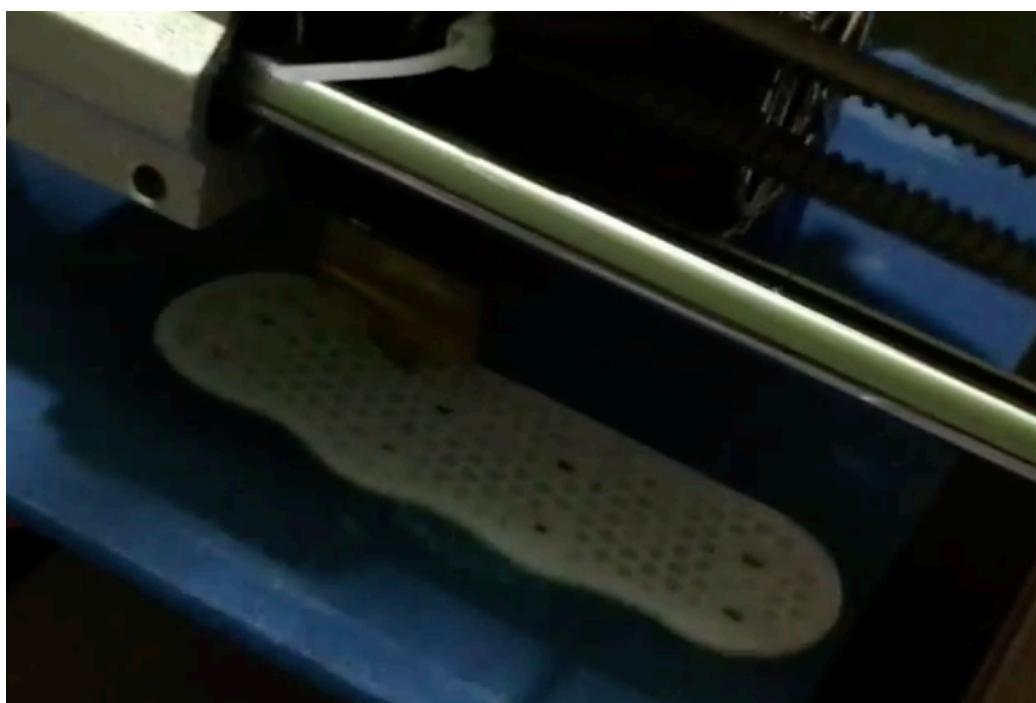


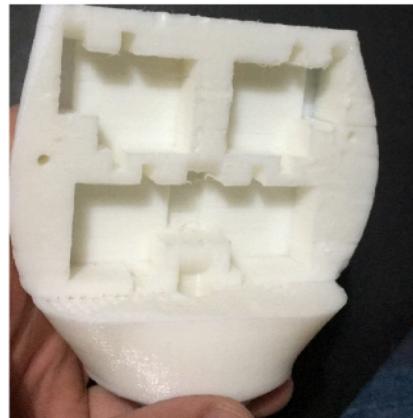
Fig. 35 - Comienzo de la impresión de la palma de la mano.

La mano fue impresa en 20 piezas, que en su mayoría fueron unidas mediante calor. El modelo cuenta con el espacio interior para montar 4 servomotores en la palma, para el movimiento de cada dedo de forma individual, y dos motores en el pulgar.

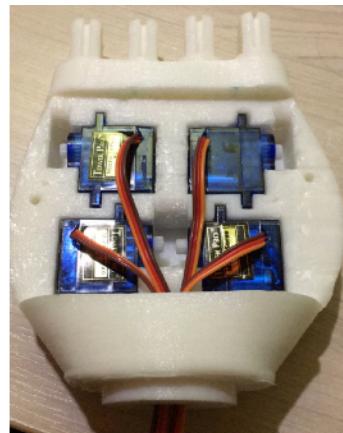
A fin de que los servomotores sean capaces de flexionar los dedos, se les acopló una rueda acanalada (también impresa en 3D), a la cual se le ataron los extremos de una tanza, que pasa por los canales internos de cada dedo, cumpliendo esta la función de tendón.

Para el servomotor que une la palma con el pulgar, se decidió utilizar uno de engranajes metálicos debido a que la fuerza requerida era mayor, por el lugar y rozamiento que supone esa posición.

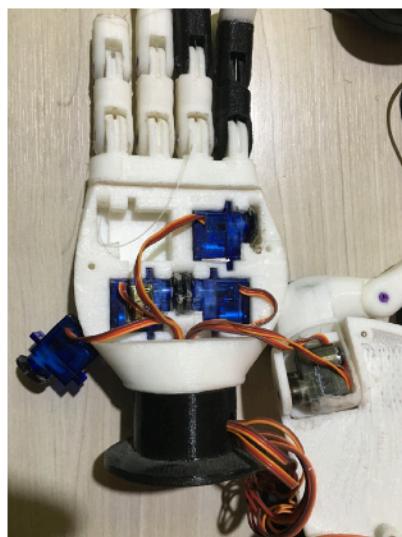
En la figura 37 se mostrarán imágenes de la mano a medida de que se fue armando, desde la primera pieza impresa hasta el resultado final cerrado.



PRIMERA PIEZA IMPRESA



PRIMERA Y SEGUNDA PIEZA CON LOS SERVOMOTORES EN POSICIÓN



MANO CASI TERMINADA CON LOS SERVOMOTORES Y LA TANZA EN POSICIÓN



MANO FINALIZADA

Fig. 36 - Comienzo de la impresión de la palma de la mano.

4.1.7. Movimiento de la mano prostética

Como se están utilizando servomotores para el movimiento de cada dedo y del pulgar, el control de la posición del mismo se realiza a través de una señal PWM. Como se tienen 6 servomotores, serán 6 las señales que deben generarse en el programa corriendo sobre la Raspberry PI.

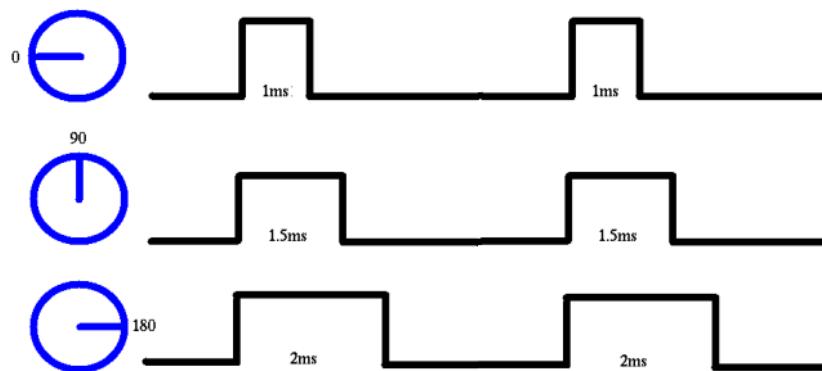


Fig. 37 - Ejemplo de señal PWM para el control de la posición de los servomotores.

Por cuestiones de comodidad en la ubicación de las salidas, se eligieron los GPIO 9, 10, 11, 17, 22 y 27 como salidas PWM.

Dentro del programa sobre NodeJS^[11], utilizando la librería pigpio, las salidas fueron declaradas de la siguiente manera:

```
var Gpio      = require('pigpio').Gpio,
menique    = new Gpio(17, {mode: Gpio.OUTPUT}),
anular     = new Gpio(27, {mode: Gpio.OUTPUT}),
medio      = new Gpio(22, {mode: Gpio.OUTPUT}),
indice     = new Gpio(10, {mode: Gpio.OUTPUT}),
palma      = new Gpio(9 , {mode: Gpio.OUTPUT}),
pulgar     = new Gpio(11, {mode: Gpio.OUTPUT});
```

Como cada dedo se controla de manera individual, se configuró cada uno también de manera individual, por lo que los valores de los extremos de movimiento de cada uno varía. A eso se le suma que no todos los servomotores están colocados en la misma posición de giro. Por ejemplo, luego del armado, las posiciones (señales PWM enviadas por la RPI) fueron las siguientes para el caso de la palma abierta:

```
menique.servoWrite(2300);
anular.servoWrite(700);
medio.servoWrite(2400);
indice.servoWrite(600);
```

```

pulgareservoWrite(2100);
palma.servoWrite(1300);

```

La función que realiza el chequeo de que deba o no realizarse el movimiento y de que gesto deba mostrarse, es llamada en cada ciclo de lectura de datos del sensor muscular.

4.2. Tiempo de trabajo y costos

El tiempo de trabajo estimado difirió de los 6 meses estimados en un principio, en mayor medida por problemas a la hora de conseguir componentes, tiempos de transporte y cantidad de ensayos superiores a los previstos.

Dentro de los costos afrontados a la hora de realizar el proyecto, los superiores no previstos fueron los de fabricado e importación de las placas necesarias.

Se muestra un cuadro de los costos en la figura 39.

| Descripción | Monto | Subtotal |
|--|---------|----------------|
| 0,5Kg PLA | \$ 200 | \$ 200 |
| 6 Servomotores (en Argentina) | \$ 600 | \$ 800 |
| ADS1294 (Finalmente no utilizado) | \$ 800 | \$ 1600 |
| Raspberry Pi Zero W (en Argentina) | \$ 800 | \$ 2400 |
| 60 Electrodos (para realización de pruebas, para el uso solo se requieren 3) | \$ 600 | \$ 3000 |
| Fabricación de 5 placas (Finalmente no utilizadas) | \$ 200 | \$ 3200 |
| Costos de envío + Costos aduaneros | \$ 2800 | \$ 6000 |
| Componentes SMD + envío | \$ 600 | \$ 6600 |
| MyoWare muscle sensor (en Argentina) | \$ 1000 | \$ 7600 |
| Varios (pegamento, tanza, tornillos) | \$ 100 | \$ 7700 |
| TOTAL DESARROLLO | | \$ 7700 |

Fig. 38 - Cuadro de costos.

En los costos no se incluyen ni la computadora ni la impresora 3D, debido a que de ellos ya se disponía anteriormente, sino debería de sumarse el costo de los mismos.

5. Resultados

Este trabajo concluye con la presentación de la mano prostética, como prototipo, explicando su funcionalidad actual.

Se proponen a su vez varias ideas para su mejora y expansión, junto a algunas pautas sobre cómo llevarlas a cabo.

5.1. Sistema final

Actualmente el sistema cumple con lo planteado en un principio, de desarrollar un sensor electromiográfico capaz de representar los datos en una pantalla y con estos datos controlar una prótesis mioeléctrica desarrollada utilizando la impresión 3D.

El objetivo principal del sistema era mantener el desarrollo en el menor costo posible, teniendo en cuenta solo los componentes utilizados la mano mioeléctrica costaría solo \$2800.- en materiales, si se compran todos en argentina, si los componentes se comprasen en china, el costo se reduciría a tan solo \$1900.- sin incluir los costos de las baterías en ninguno de los dos casos.

En este proyecto no se utilizaron baterías para la alimentación de la mano, debido a que no se disponía de baterías capaces de entregar los casi 3A de pico requeridos para el funcionamiento de los 3 servomotores. Si se alimentó la Raspberry Pi y el sensor, con una batería externa de celular, disponible en casi cualquier lugar por un precio relativamente bajo.

Este proyecto es un claro ejemplo de las posibilidades que brinda la impresión 3D a la hora de realizar prototipos de sistemas de cuerpo complejo, permitiendo en este caso, desarrollar la estructuras de los dedos con canales interiores de manera relativamente rápida y sencilla.

Por ser planteado como proyecto para finalizar la carrera en la universidad, habiendo sido posible su desarrollo gracias a los conocimientos adquiridos durante la carrera y debido a que los recursos de software utilizados son libres, se decidió subir el proyecto de la misma manera a la plataforma Github (<https://github.com/JCGuidi/ProyectoFinalElectronica2018-UTNFRBB>)

5.2. Pasos a futuro y posibles mejoras

Si bien la mano realiza los movimientos predefinidos cuando se detecta actividad muscular, la misma no cuenta con la fuerza necesaria para alguno de los movimientos que se deseé utilizar, por lo que una mejora sería la utilización de otros servomotores más fuertes, un puntapié inicial seria reemplazar los de engranaje plástico por otros de engranaje metálico. Para lograr esto, habría que rediseñar el modelo 3D del cuerpo de la mano, ya que estos servomotores son de dimensiones un poco mayores que los implementados.

Respecto a la medición de la actividad muscular, para evitar la selección de un gesto a través de la aplicación y hacer que la experiencia de uso sea más natural, podrían implementarse una mayor cantidad de sensores o uno de mayor calidad. Un sensor de mayor calidad, podría medir la diferencia de la actividad muscular, para setear diferentes movimientos para diferentes niveles, esto haría que sea más complicada la adaptación, pero brindaría una mayor independencia de la aplicación a la hora de realizar diferentes movimientos. La utilización de más de un sensor, permitiría obtener lecturas de diferentes músculos, haciendo que una combinación de excitaciones diferente produjera diferentes movimientos. Como la comunicación con los sensores es por I²C, no resultaría difícil la adaptación del sistema y la adaptación del usuario al uso no sería tan compleja como la de usar un único sensor de mayor calidad, pero haría que el sistema resultase más costoso.

Otro sensor que podría utilizarse, a fin de realizar una medición mejor es la Myo Gesture Control Armband^[28].



Fig. 39 - Myo Gesture Control Armband.

Dicha pulsera se comunica mediante Bluetooth Low Energy y gracias a su array de 8 electrodos colocados en toda su circunferencia interna, permiten leer un conjunto de músculos, haciendo que sea capaz de discernir entre diferentes gestos. Acoplar la capacidad de dicha pulsera al sistema, haría que se pudiera independizar de la aplicación web, salvo el caso en el que se quieran visualizar las mediciones. La empresa Myo brinda mucho material de apoyo y tutoriales para la inclusión de este sensor para diferentes plataformas.

Respecto a lo que es la aplicación, podría mejorarse al implementarse un cambio de interfaz en lo que respecta a configuración o bien añadirse una recomendación del threshold y no tener que ser seteado por el usuario teniendo en cuenta los valores medidos desde que se conecta la aplicación.

Otra mejora sería desarrollar la misma aplicación para el SO Android, a fin de que la cantidad de usuarios a las que pudiese llegar sea mayor.

Referencias Bibliográficas

1. Desarrollo de dispositivos e-Health de bajo coste para Raspberry Pi - Curso 2013/2014 - Xavier Gallofré Nieva Jesús F. López San José Álvaro Pérez Liaño.
2. Sistema de Adquisición y Procesamiento Inteligente de Señales Biológicas - Norberto Scarone, Damián Marasco, Nicolás Castro, Gustavo Monte.
3. Datasheet ads1298 - Texas Instruments
4. Datasheet ATmega328/P - Atmel
5. Datasheet MyoWare™ Muscle Sensor (AT-04-001) - Advancer Technologies
6. The Swift Programming Language - Apple
7. BCM2835 ARM Peripherals. - Broadcom Corporation. Broadcom Europe Ltd. 406 Science Park Milton Road Cambridge CB4 0WW, 2012.
8. <https://www.thingiverse.com/thing:1103992>
9. <https://www.raspberrypi.org/>
10. <https://www.raspberrypi.org/blog/introducing-raspberry-pi-hats/>
11. <https://nodejs.org/es/about/>
12. <https://github.com/noble/bleno>
13. <https://www.npmjs.com/package/i2c>
14. <https://www.mayoclinic.org/es-es/tests-procedures/emg/pac-20393913>
15. <https://www.sparkfun.com/products/13723>
16. <http://raspi.tv/2017/how-much-power-does-pi-zero-w-use>
17. <https://www.amputee-coalition.org/resources/spanish-history-prosthetics/>
18. <https://atomiclab.org>
19. <http://www.vocativ.com/money/industry/prosthetic-boom-3d-printed-mind-controlled-limbs/index.html>
20. <http://www.monografias.com/trabajos96/bioingenieria-protesis-bionicas-manos/bioingenieria-protesis-bionicas-manos.shtml>
21. <https://solidgeargroup.com/bluetooth-ble-el-conocido-desconocido?lang=es>
22. <https://www.bluetooth.com/specifications/gatt/generic-attributes-overview>
23. [https://es.wikipedia.org/wiki/Swift_\(lenguaje_de_programaci%F3n\)](https://es.wikipedia.org/wiki/Swift_(lenguaje_de_programaci%F3n))
24. <https://developer.apple.com/swift/>
25. <https://es.wikipedia.org/wiki/Xcode>
26. <https://openbionics.com/>
27. <http://exiii-hackberry.com>
28. <https://www.myo.com>

Anexo I

MyoWare Muscle Sensor

Anexo II

Servidor GATT NodeJS

```

// runBLE.sh
//
// Created by Juan Cruz Guidi.
// Copyright © 2018 Juan Cruz Guidi. All rights reserved.
//

sudo service bluetooth stop
sudo hciconfig hci0 up
sudo node bleno/examples/echo/main.js


//


// main.js
//
// Created by Juan Cruz Guidi.
// Copyright © 2018 Juan Cruz Guidi. All rights reserved.
//


var bleno = require('../..');

var BlenoPrimaryService = bleno.PrimaryService;
var EchoCharacteristic = require('./coreGPIO');

console.log('bleno - MAN0');

bleno.on('stateChange', function(state) {
  console.log('on -> stateChange: ' + state);

  if (state === 'poweredOn') {
    bleno.startAdvertising('MAN0', ['ec00']);
  } else {
    bleno.stopAdvertising();
  }
});

bleno.on('advertisingStart', function(error) {
  console.log('on -> advertisingStart: ' + (error ? 'error ' + error : 'success'));
}

if (!error) {
  bleno.setServices([
    new BlenoPrimaryService({
      uuid: 'ec00',
      characteristics: [
        new EchoCharacteristic()
      ]
    })
  ]);
}
});
```

```

// coreGPIO.js
//
// Created by Juan Cruz Guidi.
// Copyright © 2018 Juan Cruz Guidi. All rights reserved.
//

var util = require('util');
var bleno = require('../');
var i2c = require('i2c');
var address = 0x04;
var wire = new i2c(address, {device: '/dev/i2c-1'});
var readValue = 0;
var myVar = 0;
var BlenoCharacteristic = bleno.Characteristic;
var threshold = 100;
var configuration = 5;

var Gpio = require('pigpio').Gpio,
    menique = new Gpio(17, {mode: Gpio.OUTPUT}),
    anular = new Gpio(27, {mode: Gpio.OUTPUT}),
    medio = new Gpio(22, {mode: Gpio.OUTPUT}),
    indice = new Gpio(10, {mode: Gpio.OUTPUT}),
    palma = new Gpio(9, {mode: Gpio.OUTPUT}),
    pulgar = new Gpio(11, {mode: Gpio.OUTPUT}),
    led = new Gpio(5, {mode: Gpio.OUTPUT}),
    pulseWidth = 1000;

led.digitalWrite(1);

//Aregar tiempos, control posicion blablabla
function palmaAbierta() {
    menique.servoWrite(2400);
    anular.servoWrite(600);
    medio.servoWrite(2400);
    indice.servoWrite(770);
    pulgar.servoWrite(2400);
    palma.servoWrite(1420);
}
function fYou() {
    menique.servoWrite(600);
    anular.servoWrite(2400);
    medio.servoWrite(2400);
    indice.servoWrite(2400);
    pulgar.servoWrite(2100);
    setTimeout(function(){palma.servoWrite(1050);}, 200);
    setTimeout(function(){pulgar.servoWrite(1925);}, 200);
}
function paz() {
    menique.servoWrite(600);
    anular.servoWrite(2400);
    medio.servoWrite(2400);
    indice.servoWrite(770);
    setTimeout(function(){palma.servoWrite(1080);}, 200);
    setTimeout(function(){pulgar.servoWrite(1600);}, 200);
}
function senalar() {
    menique.servoWrite(600);
    anular.servoWrite(2400);
    medio.servoWrite(600);
    indice.servoWrite(770);
    setTimeout(function(){palma.servoWrite(1080);}, 200);
    setTimeout(function(){pulgar.servoWrite(1780);}, 200);
}
function queonda() {
    menique.servoWrite(2400);
    anular.servoWrite(2400);
    medio.servoWrite(600);
    indice.servoWrite(2400);
    pulgar.servoWrite(2400);
}

```

```

    palma.servoWrite(1420);
}
function cerrada(){
    menique.servoWrite(600);
    anular.servoWrite(2400);
    medio.servoWrite(600);
    indice.servoWrite(2400);
    pulgar.servoWrite(2100);
    setTimeout(function(){palma.servoWrite(1050);}, 200);
    setTimeout(function(){pulgar.servoWrite(1925);}, 200);
}

function myFunc() {
    wire.read(1, function(err, res) {
        readValue = res[0];
        if (readValue > threshold) {
            switch (configuration) {
                case 1:
                    setTimeout(fYou, 500);
                    break;
                case 2:
                    setTimeout(paz, 500);
                    break;
                case 3:
                    setTimeout(senalar, 500);
                    break;
                case 4:
                    setTimeout(queonda, 500);
                    break;
                case 5:
                    cerrada()
                    break;
                default:
                    palmaAbierta()
            }
        } else {
            palmaAbierta();
        }
    });
}

var EchoCharacteristic = function() {
    EchoCharacteristic.super_.call(this, {
        uuid: 'ec0e',
        properties: ['read', 'write', 'notify'],
        value: null
    });

    this._value = new Buffer(0);
    this._updateValueCallback = null;
};

util.inherits(EchoCharacteristic, BlenoCharacteristic);

EchoCharacteristic.prototype.onReadRequest = function(offset, callback) {
    console.log('EchoCharacteristic - onReadRequest: value = ' + this._value);
    callback(this.RESULT_SUCCESS, this._value);
};

EchoCharacteristic.prototype.onWriteRequest = function(data, offset,
withoutResponse, callback) {
    this._value = data;

    if (data < 10) {
        switch (Math.floor(data)) {
            case 1:
                configuration = 1;
                break;
            case 2:

```

```

        configuration = 2;
        break;
    case 3:
        configuration = 3;
        break;
    case 4:
        configuration = 4;
        break;
    case 5:
        configuration = 5;
        break;
    default:
        configuration = 5;
    }
} else if (data > 250) {
    switch (Math.floor(data/10000) ) {
        case 1:
            menique.servoWrite(data%10000);
            break;
        case 2:
            anular.servoWrite(data%10000);
            break;
        case 3:
            medio.servoWrite(data%10000);
            break;
        case 4:
            indice.servoWrite(data%10000);
            break;
        case 5:
            pulgar.servoWrite(data%10000);
            break;
        case 6:
            palma.servoWrite(data%10000);
            break;
        default:
            console.log("awantaa");
            //👉: 2300 - 700 - 2400 - 600 - p1300 - b1400
            menique.servoWrite(2300);
            anular.servoWrite(700);
            medio.servoWrite(2400);
            indice.servoWrite(600);
            pulgar.servoWrite(2100);
            palma.servoWrite(1300);
    }
} else {
    threshold = data
}

console.log('EchoCharacteristic - onWriteRequest: value = ' + this._value);

if (this._updateValueCallback) {
    console.log('EchoCharacteristic - onWriteRequest: notifying');
}

callback(this.RESULT_SUCCESS);
};

EchoCharacteristic.prototype.onSubscribe = function(maxValueSize,
updateValueCallback) {
    console.log('EchoCharacteristic - onSubscribe');
    myVar = setInterval(myFunc, 100);
    this.changeInterval = setInterval(function() {
        var data = new Buffer(2);
        data.writeUInt16LE(readValue, 0);
        //console.log('IndicateOnlyCharacteristic update value:' + readValue);
        updateValueCallback(data);
    }.bind(this), 100);

    this._updateValueCallback = updateValueCallback;
}

```

```
};

EchoCharacteristic.prototype.onUnsubscribe = function() {
    console.log('EchoCharacteristic - onUnsubscribe');
    clearInterval(myVar);
    if (this.changeInterval) {
        clearInterval(this.changeInterval);
        this.changeInterval = null;
    }
    this._updateValueCallback = null;
};

module.exports = EchoCharacteristic;
```

Anexo III

Aplicación iOS

```

// ViewController.swift
// chart
//
// Created by Juan Cruz Guidi.
// Copyright © 2018 Juan Cruz Guidi. All rights reserved.
//

import UIKit
import Charts
import CoreBluetooth

class ViewController: UIViewController, CBCentralManagerDelegate, CBPeripheralDelegate, ChartViewDelegate {

    //-----Chart-----
    @IBOutlet weak var lineChartView: LineChartView!

    @IBOutlet weak var estado: UILabel!
    @IBOutlet weak var onOffBut: UIButton!
    @IBOutlet weak var selectedOption: UILabel!

    var doubleDouble = [[Double](), [Double]()
    var timer = Timer()
    var dataEntries: [ChartDataEntry] = []
    let data = LineChartData()
    var frecMin = 0

    //-----Bluetooth-----

    var manager : CBCentralManager!
    var miBT : CBPeripheral!
    var miCharacteristic : CBCharacteristic!
    var isConected = false
    var isRunning = false

    //-----Data to send-----

    let onData: Data = "1".data(using: .utf8)!
    let offData: Data = "0".data(using: .utf8)!

    override func viewDidLoad() {
        super.viewDidLoad()

        lineChartView.delegate = self
        manager = CBCentralManager(delegate: self, queue: nil)
        doubleDouble = [[0,0]]
        for i in 0..

```

```

func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNumber) {
    if peripheral.name == "raspberrypi" {
        self.miBT = peripheral
        self.miBT.delegate = self
        manager.stopScan()
        manager.connect(miBT, options: nil)
    }
}

func peripheral(_ peripheral: CBPeripheral, didDiscoverServices error: Error?) {
    if let servicePeripheral = peripheral.services as [CBService]! {
        for service in servicePeripheral {
            peripheral.discoverCharacteristics(nil, for: service)
        }
    }
}

func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService, error: Error?) {
    if let characterArray = service.characteristics as [CBCharacteristic]! {

        for cc in characterArray {
            if(cc.uuid.uuidString == "EC0E") {
                miCharacteristic = cc
                peripheral.writeValue("0".data(using: .utf8)!, for: cc, type: CBCharacteristicWriteType.withResponse)
                peripheral.readValue(for: cc)
            }
        }
    }
}

func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic, error: Error?) {
    if (characteristic.uuid.uuidString == "EC0E") {
        let data = characteristic.value! as Data
        let value = UInt32(littleEndian: data.withUnsafeBytes { $0.pointee })
        addPoint(value: Int(value))
    }
}

func centralManager(_ central: CBCentralManager, didConnect peripheral: CBPeripheral) {
    isConected = true
    estado.text = "Conectado"
    peripheral.delegate = self
    peripheral.discoverServices(nil)
}

func peripheral(_ peripheral: CBPeripheral, didReadRSSI RSSI: NSNumber, error: Error?) {
    print(RSSI.intValue)
}

func centralManager(_ central: CBCentralManager, didDisconnectPeripheral peripheral: CBPeripheral, error: Error?) {
    isConected = false
}

```

```

        estado.text = "Desconectado"
        manager.scanForPeripherals(withServices: nil, options: nil)
    }

func centralManagerDidUpdateState(_ central: CBCentralManager) {
    switch central.state {
    case .poweredOff:
        //msg = "BLE Off"
        estado.text = "El Bluetooth está apagado..."
    case .poweredOn:
        //msg = "BLE On"
        manager.scanForPeripherals(withServices: nil, options: nil)
    case .unsupported:
        print("unSupported")
    default:
        print("default")
    }
}

func peripheral(_ peripheral: CBPeripheral, didUpdateNotificationStateFor
characteristic: CBCharacteristic, error: Error?) {

    if (error != nil) {
        print("Error changing notification state: \(error ?? "error" as!
Error)")
    } else {
        peripheral.readValue(for: miCharacteristic)
    }
}

func addPoint(value: Int) -> Void {
    freqMin += 1
    let thisY = Double(value)
    let dataEntry = BarChartDataEntry(x: Double(freqMin), y: Double(thisY))

    if freqMin > 2500 {
        self.lineChartView.data?.removeEntry(xValue: Double(freqMin - 2501),
dataSetIndex: 0)
    }

    self.lineChartView.data?.addEntry(dataEntry, dataSetIndex: 0)
    self.lineChartView.notifyDataSetChanged()
    self.lineChartView.setVisibleXRangeMaximum(250)
    self.lineChartView.moveViewToX(Double(freqMin))
}

override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()
    // Dispose of any resources that can be recreated.
}

func chartValueSelected(_ chartView: ChartViewBase, entry: ChartDataEntry,
highlight: Highlight) {
    self.lineChartView.chartDescription?.text = "(\(entry.x) , \(entry.y))"
}

@IBAction func onOffAction(_ sender: Any) {

    if isConnected {
        if isRunning {
            onOffBut.setTitle("On", for: .normal)
            miBT.setNotifyValue(false, for: miCharacteristic)
            self.lineChartView.isUserInteractionEnabled = true
            isRunning = false
        } else {
            onOffBut.setTitle("Off", for: .normal)
            miBT.writeValue(onData, for: miCharacteristic, type:
CBCharacteristicWriteType.withResponse)
            miBT.setNotifyValue(true, for: miCharacteristic)
        }
    }
}

```

```

        self.lineChartView.isUserInteractionEnabled = false
        isRunning = true
    }
}

@IBAction func openHand(_ sender: Any) {
    if isConected {
        self.miBT.writeValue("5".data(using: .utf8)!, for:
self.miCharacteristic, type: CBCharacteristicWriteType.withResponse)
        selectedOption.text = "👉"
    }
}

@IBAction func fYou(_ sender: Any) {
    if isConected {
        self.miBT.writeValue("1".data(using: .utf8)!, for:
self.miCharacteristic, type: CBCharacteristicWriteType.withResponse)
        selectedOption.text = "👉"
    }
}

@IBAction func pointAction(_ sender: Any) {
    if isConected {
        self.miBT.writeValue("3".data(using: .utf8)!, for:
self.miCharacteristic, type: CBCharacteristicWriteType.withResponse)
        selectedOption.text = "👉"
    }
}

@IBAction func coolAction(_ sender: Any) {
    if isConected {
        self.miBT.writeValue("4".data(using: .utf8)!, for:
self.miCharacteristic, type: CBCharacteristicWriteType.withResponse)
        selectedOption.text = "👉"
    }
}

@IBAction func configurarValor(_ sender: Any) {
    if isConected {
        let alertVC = UIAlertController(title: "Ingresar Valor", message:
"Para cambiar la configuración", preferredStyle: .alert)

        alertVC.addTextField { (textField) in
            textField.keyboardType = .numberPad
            textField.placeholder = "Valor"
        }

        let submitAction = UIAlertAction(title: "Cambiar", style: .default,
handler: {
            (alert) -> Void in

            let textField = alertVC.textFields![0] as UITextField
            print("Email -- \(textField.text!)")

            self.miBT.writeValue((textField.text ?? "").data(using: .utf8)!, for:
self.miCharacteristic, type: CBCharacteristicWriteType.withResponse)
        })

        alertVC.addAction(submitAction)
        alertVC.view.tintColor = UIColor.black
        present(alertVC, animated: true)
    } else {
        let alert = UIAlertController(title: "No Conectado", message:
"Conectese a la mano para poder configurarla", preferredStyle: .alert)
    }
}

```

```
        alert.addAction(UIAlertAction(title: "Ok", style: .default, handler:  
nil))  
        self.present(alert, animated: true, completion: nil)  
    }  
}  
  
@IBAction func guardarGraph(_ sender: Any) {  
    if !isRunning {  
  
UIImageWriteToSavedPhotosAlbum(self.lineChartView.getChartImage(transparent:  
false)!, self, #selector(image(_:didFinishSavingWithError:contextInfo:)), nil)  
    }  
  
    func image(_ image: UIImage, didFinishSavingWithError error: NSError?,  
contextInfo: UnsafeRawPointer) {  
        if let error = error {  
            // we got back an error!  
            let ac = UIAlertController(title: "Error", message:  
error.localizedDescription, preferredStyle: .alert)  
            ac.addAction(UIAlertAction(title: "OK", style: .default))  
            present(ac, animated: true)  
        } else {  
            let ac = UIAlertController(title: "Guardada!", message: "La imagen  
del gráfico fue guardada", preferredStyle: .alert)  
            ac.addAction(UIAlertAction(title: "OK", style: .default))  
            present(ac, animated: true)  
        }  
    }  
}
```