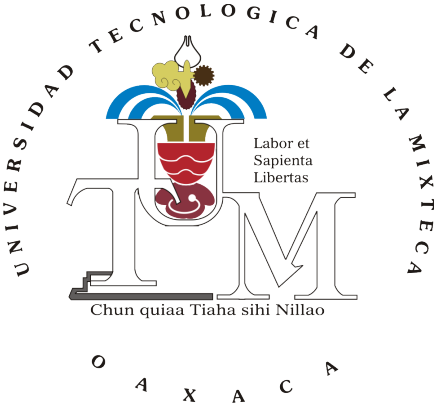
UNIVERSIDAD TECNOLÓGICA DE LA MIXTECA



DESARROLLO DE UN PROTOTIPO DE PRÓTESIS ACTIVA DE MIEMBRO SUPERIOR UTILIZANDO SEÑALES MIOELÉCTRICAS

TESIS

PARA OBTENER EL TÍTULO DE
INGENIERO EN MECATRÓNICA

PRESENTA:

LEOBARDO ELÍ SÁNCHEZ VELASCO

DIRECTOR DE TESIS:

DR. MANUEL ARIAS MONTIEL

CO-DIRECTOR DE TESIS:

DR. ENRIQUE GUZMÁN RAMÍREZ

HUAJUAPAN DE LEÓN, OAXACA, MÉXICO, ABRIL DEL 2019

Índice

Índice de figuras

Índice de Tablas

Capítulo 1

Introducción

La mano es una de las partes del cuerpo humano más complejas y versátiles ya que se requiere una interacción perfecta de nervios, tendones, huesos, músculos y articulaciones que permitan realizar tareas cotidianas de forma natural. Uno de los grandes retos para la tecnología médica es reproducir en una prótesis tantos movimientos y funciones de la mano humana como sea posible [1].

Un factor limitante en el desarrollo de prótesis activas es la dificultad de encontrar un número adecuado de fuentes de control para el número de grados de libertad que requiera el sistema. Ésto es un impedimento muy grande para el desarrollo de prótesis, sin embargo es de suma importancia desarrollar mejores mecanismos y actuadores ya que el control sería poco útil si no hay un sistema efectivo que controlar [2].

Uno de los principales objetivos de la investigación de prótesis activas en extremidades superiores es el de diseñar brazos artificiales y reemplazos de manera que sean capaces de realizar movimientos a altas velocidades y con la suficiente fuerza de agarre.

Las prótesis mioeléctricas (controladas mediante los campos eléctricos generados como producto de las contracciones musculares) brindan el mayor grado de rehabilitación que puedan ofrecer las prótesis activas. Desafortunadamente, los componentes y las técnicas de interfaz utilizadas actualmente para la fabricación de estos sistemas aún están en desarrollo.

En este trabajo de tesis se presenta el desarrollo de un prototipo de prótesis activa de miembro superior, la cual es controlada con señales mioeléctricas obtenidas de un brazalete comercial. El prototipo parte de un diseño mecánico previamente desarrollado y está orientado a realizar gestos de agarre comunes en la mano humana.

1.1. Antecedentes

En esta sección se presenta una breve reseña histórica del desarrollo de las prótesis, así como una clasificación de los tipos de prótesis funcionales que existen en la actualidad y algunas prótesis mioeléctricas disponibles en el mercado.

1.1.1. Prótesis

Las prótesis son aparatos externos utilizados para reemplazar total o parcialmente un miembro deficiente o ausente [3] y su implementación ha existido desde las primeras culturas.

En los últimos años, se ha puesto mucho énfasis en desarrollar miembros artificiales que se vean y se muevan como si fueran los miembros humanos reales, sin embargo la mayoría de prótesis de miembro superior se limitan al cierre y apertura del dispositivo prensil.

Los avances en la comprensión de la biomecánica del cuerpo humano, con el trabajo combinado de médicos y de ingenieros, el desarrollo de nuevos plásticos, y el uso de las técnicas de Diseño y Manufactura Asistidos por Computadora han contribuido en el desarrollo de miembros artificiales más realistas [4].

En la actualidad, las prótesis activas de miembro superior se pueden clasificar en 5 tipos de acuerdo al tipo de control que se utilice:

Prótesis mecánicas

Actualmente éstas prótesis se controlan con la fuerza del cuerpo del usuario de forma indirecta por medio de movimientos musculares, ya sea del hombro o del omóplato (hueso ubicado en la parte superior de la espalda) con los cuales diferentes segmentos de arneses son tirados, activando los componentes de la prótesis que ofrecen alguna funcionalidad de agarre [5].

Su funcionamiento se basa en la extensión de una liga por medio de un arnés para la apertura o cierre del dispositivo prensor, y se efectúa solo con la relajación del músculo gracias a un resorte que brinda una fuerza de presión ó pellizco. Estos elementos generalmente se recubren con un guante para dar una apariencia más estética, sin embargo se limita al agarre de objetos relativamente grandes y redondos ya que el guante estorba al querer sujetar objetos pequeños. En la Figura 1.1 se puede observar una prótesis mecánica con sus arneses y la liga que transmite el movimiento. El tamaño de la prótesis y el número de ligas que se requiera

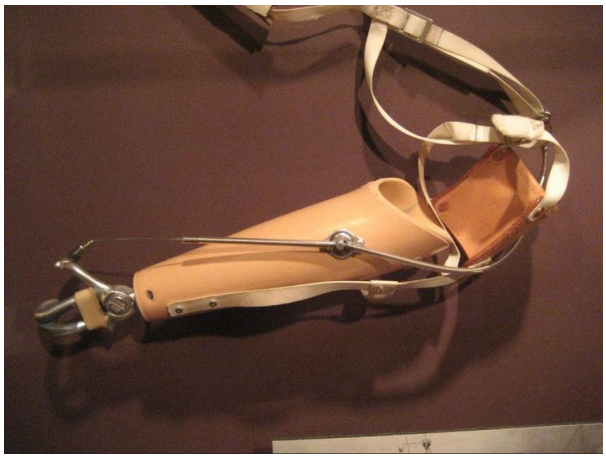


Figura 1.1: Prótesis mecánica de gancho.

dependen de la fuerza y el material para su fabricación, que varían de acuerdo a las necesidades

de cada persona. Dado que estas prótesis son accionadas por el cuerpo, es necesario que el usuario posea al menos un movimiento general de expansión del pecho, depresión y elevación del hombro, abducción y aducción escapular (movimientos del hueso escrapular ubicado en la parte posterior del tórax a ambos lados de la columna vertebral, que lo alejan o lo acercan a ella) o flexión en la articulación del hombro (glenohumeral) [6].

Prótesis eléctricas

Son controladas de diversas maneras, usando servo-controles, control con botones pulsantes o interruptores. En ciertas ocasiones se combinan estas formas para mejorar su funcionalidad. Se usa un socket que es un dispositivo intermedio entre la prótesis y el muñón, logrando la suspensión de éste por medio de sistemas de succión. Su principal desventaja es su alto costo de adquisición y reparación, así como su peso y el cuidado que se debe tener con su exposición ante medios húmedos [6] [7].

En la Figura 1.2 se muestra la mano eléctrica 2000 ® diseñada para niños.

Figura 1.2: Prótesis eléctrica 2000® para niños[8].

Prótesis neumáticas

Las prótesis neumáticas son desarrolladas contemporáneamente con las eléctricas y hacen uso de aire a presión obtenido por medio de un compresor para activar su función. Su ventaja principal es proporcionar una gran fuerza y rapidez de movimientos; sus desventajas principales son los dispositivos que se implementan para su control y funcionamiento ya que son relativamente grandes y su mantenimiento es costoso y complejo [7]. En la Figura 1.3 se presenta la mano neumática Shadow® desarrollada por la empresa Shadow Robot Company© a principios del año 2000.

Prótesis mioeléctricas

Las prótesis mioeléctricas (también llamadas prótesis biónicas) comenzaron a desarrollarse en el año de 1960. Son controladas mediante señales electromiográficas (señales EMG) que se encargan de activar y controlar el movimiento funcional de la prótesis [6] [9] [10].

Las señales EMG son captadas a través de electrodos que entran en contacto con la piel en la zona donde se encuentra el músculo que generará la señal. Una vez captada, se amplifica y se procesa en un controlador que active y desactive los motores para producir movimientos y funcionalidad.

Figura 1.3: Prótesis neumática Shadow® [11].

Este tipo de prótesis tiene la ventaja de que sintetizan el mejor aspecto estético, tienen gran fuerza y velocidad, así como muchas posibilidades de combinación y ampliación. Sólo requieren que el usuario flexione sus músculos para operarla, a diferencia de las prótesis accionadas por el cuerpo que requieren el movimiento general del cuerpo lo que permite el mayor grado de rehabilitación en miembros artificiales [5] [6].

Como desventaja, se presenta el uso de baterías ya que requieren mantenimiento para recargarlas y eventualmente para reemplazarlas. El peso de este tipo de prótesis puede llegar a ser mayor que las prótesis mecánicas debido a los motores y baterías, además de que una prótesis accionada por electricidad proporciona un mayor nivel de tecnología, pero a un mayor costo [5] [6] [9]. En la Figura 1.4 se puede observar una prótesis mioeléctrica con los electrodos utilizados para adquirir las señales EMG.

Figura 1.4: Prótesis mioeléctrica [12].

Prótesis híbridas

Las prótesis híbridas son aquellas que combinan el modo de controlar el sistema, utilizando la acción del cuerpo y el accionamiento por electricidad en una sóla. La mayor parte de este tipo de prótesis son utilizadas para cubrir amputaciones por arriba del codo (transhumerales) ya que frecuentemente se utiliza un codo accionado por el cuerpo y un dispositivo terminal

como un gancho o mano, controlado por señales electromiográficas [13]. La Figura 1.5 muestra un ejemplo de prótesis híbrida.

Figura 1.5: Prótesis híbrida [12].

1.1.2. Prótesis mioeléctricas existentes

En el desarrollo de prótesis activas, las mioeléctricas son las más populares debido a que con este sistema se logra el control con movimientos más naturales y brindan un mayor grado de rehabilitación en comparación con otros tipos de prótesis convencionales [14].

Los sistemas protésicos de miembro superior mioeléctricos que pueden ser encontrados en el mercado son contados, ya que la mayoría aún están en desarrollo. A continuación se presentan algunos de ellos.

Sistema de mano protésica MyoFacil®

El sistema de mano protésica MyoFacil® (mostrado en la Figura 1.6) es desarrollado por la empresa de origen alemán Ottobock©, la cual comenzó con el desarrollo de prótesis mioeléctricas en los años 60. Consiste en una adaptación protésica básica, de un grado de libertad, que permite abrir y cerrar la mano mediante señales musculares, especialmente para realizar actividades en el hogar u oficina. La prótesis contiene un interruptor que permite apagarla para hacer uso de ella de manera pasiva [15].

Figura 1.6: Prótesis MyoFacil [15].

Prótesis Michelangelo

También desarrollada por la empresa Ottobock©, la mano mioeléctrica Michelangelo® (Figura 1.7) fue desarrollada a principios del año 2000 y pretende proporcionar más funciones naturales debido a sus opciones de agarre predeterminados, además de contar con una muñeca que se puede flexionar, extender y rotar hacia dentro y hacia fuera. Puede ejercer una fuerza de entre 6 y 7 kg la cual se puede ajusar de acuerdo a las necesidades de agarre, dependiendo del objeto a manipular [16].

Figura 1.7: Prótesis Michelangelo® [16].

Prótesis biónica I-Limb®

I-Limb® es una prótesis mioeléctrica creada en Irlanda por la compañía Touch Bionics© y lanzada al mercado en el 2007. Esta prótesis utiliza electrodos colocados en la piel en dos sitios del músculo preseleccionados, de manera que cuando se contraen dichos músculos la mano se mueve de acuerdo al tipo de agarre programado.

La prótesis contiene posiciones de agarre diferentes que se activan de acuerdo a la actividad muscular del paciente. Es por esto que la cantidad de agarres dependerá de la fuerza y control que el usuario tenga con sus músculos remanentes, de manera que con algo de práctica puedan agregarse más posiciones de agarre [17].

En la Figura 1.8 se muestra la prótesis I-limb®.

Figura 1.8: Prótesis I-Limb [17].

Prótesis Bebionic®

La prótesis creada por la empresa Steeper de Reino Unido, actualmente es desarrollada por la compañía Ottobock© tras su adquisición en el 2017 y su nombre es Bebionic® (Figura 1.9). Como característica principal presenta un motor en cada dedo permitiendo un mejor agarre, siendo ligera y cómoda.

Tiene un total de 14 formas de sujeción disponibles, lo que permite realizar gran número de quehaceres diarios. Tiene posiciones seleccionables de pulgar y está construido con materiales avanzados y duraderos, que permiten soportar hasta 45kg [18].

Figura 1.9: Prótesis Bebionic® [18].

Prótesis biónica de Probionics®

La prótesis desarrollada entre los años 2008 y 2012 por la compañía mexicana Probionics® cuenta con dos grados de libertad que permiten realizar movimientos de apertura/cierre de la mano y girar la muñeca, lo que permite realizar acciones básicas para su uso diario en el hogar. El sistema es controlado mediante contracciones musculares del miembro remanente y trabaja con una batería que le permite al sistema funcionar por dos días [19]. En la Figura 1.10 se muestra el sistema colocado en un paciente.

Figura 1.10: Prótesis desarrollada por Probionics® [19].

A continuación se presenta un proyecto cuyo diseño mecánico se utilizará como base para el desarrollo de este trabajo.

Mano de 6 grados de libertad de código abierto

En el año 2016 se publicó el artículo titulado "Design and Fabrication of a Six Degree-of-Freedom Open Source Hand" [20] en el cual se presenta un proyecto que tiene como objetivo desarrollar una mano robótica que permita la implementación y pruebas de algoritmos de control, que faciliten el desarrollo de un exitoso e intuitivo método de control para prótesis mioeléctricas.

La mano (mostrada en la Figura 1.11) es capaz de producir cualquier postura prensil común, además de ejecutar tareas más complejas que involucren movimientos independientes de los dedos. El diseño del prototipo contiene 6 grados de libertad, cuenta con un actuador que proporciona un movimiento de flexión en cada dedo a excepción del pulgar, el cual contiene un actuador adicional que le permite hacer un movimiento de rotación [20].

Adicionalmente los autores han decidido hacer el diseño de código abierto, proporcionando los archivos CAD, planes de manufactura e instrucciones de ensamble con el fin de contribuir de manera más completa en la investigación y desarrollo de prótesis. El diseño está restringido en su mayor parte por los costos y su enfoque de código abierto. La mayoría de los componentes de la mano están diseñados para ser manufacturados fácilmente usando una impresora 3D, evitando procesos más caros y laboriosos [20].

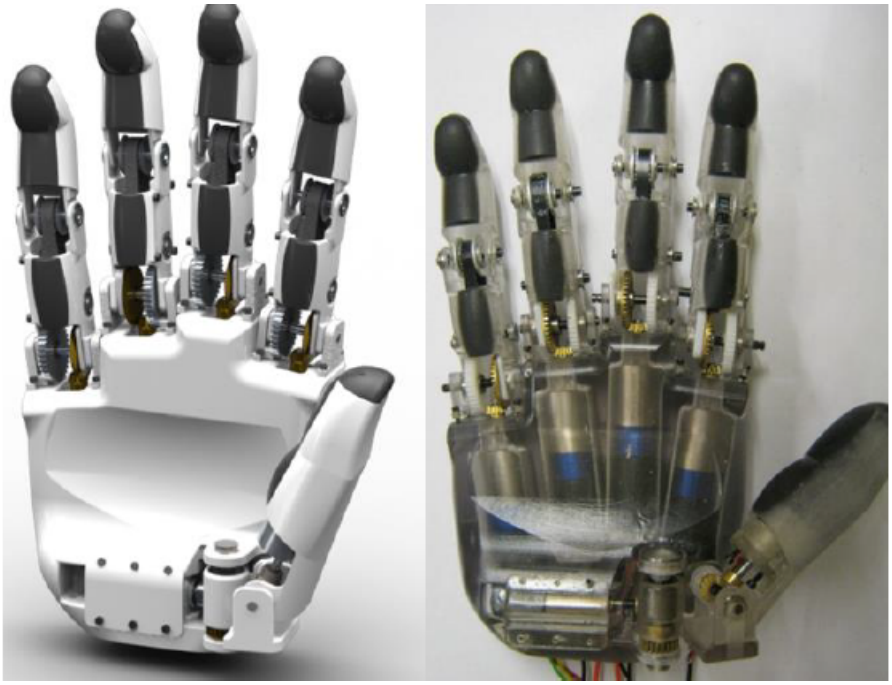


Figura 1.11: Mano de 6 grados de libertad de código abierto [20].

Debido a las características de la mano que pueden ser útiles en el desarrollo de este proyecto, se tomará como base el diseño de la mano de 6 grados de libertad (6 GDL) para controlarla con un sistema mioeléctrico, modificando algunos mecanismos para facilitar su fabricación y hacer económicamente más accesible la adquisición de algunos de sus componentes, particularmente los actuadores y engranes.

1.2. Planteamiento del problema

Las prótesis mioeléctricas de miembro superior han sido desarrolladas para realizar diversos gestos permitiendo al usuario interactuar de manera más sencilla con los objetos a su alrededor. Sin embargo, en la mayoría de prótesis electromiográficas (omitiendo aquellas que requieren de una cirugía en el paciente para poder operarla) sólo se utilizan dos sensores EMG como entradas de control del sistema, de manera que su modo de operación se ve limitado por este número de sensores.

Los procedimientos de control comúnmente aplicados en dichos sistemas le permiten a la prótesis realizar movimientos predeterminados, como abrir o cerrar un dispositivo prensor (para prótesis de 1 grado de libertad) o cambiar entre posturas predefinidas (dispositivos con 5 o 6 grados de libertad).

En este proyecto se propone implementar una forma alternativa de controlar una prótesis mioeléctrica aumentando el número de sensores EMG del sistema. Debido a ésto se requerirá de un prototipo de prótesis versátil capaz de reproducir diversos movimientos, un sistema de control para manipular los movimientos de la mano y un sistema para la obtención y procesamiento de la información obtenida por los sensores electromiográficos a utilizar.

La mano de 6 grados de libertad (GDL) de código abierto [20] fue realizada para ejecutar diversos gestos de la mano humana, por lo que se tomará como referencia para el desarrollo del proyecto. Sin embargo, la mano robótica carece de elementos que permitan controlar sus movimientos (como encoders y drivers) por lo que será necesario modificar su diseño para adaptarlo a los nuevos elementos y añadir versatilidad al dedo pulgar.

1.3. Justificación

El desarrollo de prótesis con sensores electromiográficos adicionales brindará más señales de control al sistema y se podrán implementar distintos clasificadores para reconocer diferentes actividades musculares, ampliando los posibles modos de operación de las prótesis.

En el prototipo se podrán implementar diferentes formas de controlar la mano y de analizar los datos obtenidos con el brazalete, por lo que facilitará investigaciones de diversos algoritmos de control y de procesamiento de datos para controlar la prótesis.

Gracias a la implementación del diseño de la mano de 6 GDL de código abierto se podrán generar diversos gestos de agarre, permitiendo al usuario realizar diversas tareas que con otras prótesis no podría.

1.4. Objetivos

1.4.1. Objetivo general

Desarrollar un prototipo de prótesis de miembro superior activa y controlarlo mediante señales electromiográficas con el fin de obtener un sistema capaz de proporcionar distintos tipos de agarre prensil humano.

1.4.2. Objetivos específicos

Para acatar las expectativas del objetivo general se deberán cumplir los siguientes objetivos específicos:

- ✓ Realizar modificaciones al diseño de referencia para adaptar los nuevos elementos, mejorar la versatilidad del dedo pulgar y reducir los costos de adquisición de algunos componentes.
- ✓ Fabricar un prototipo de mano robótica basada en el diseño de referencia y en el punto anterior.
- ✓ Fabricar un socket para montar la mano.
- ✓ Obtener señales EMG mediante un brazalete mioeléctrico comercial y procesarlas para controlar la mano sin la dependencia de una PC.
- ✓ Desarrollar un sistema que controle los actuadores de la mano para generar los 6 de los gestos más comunes en la mano humana (Agarre cilíndrico, de punta, de tipo gancho, palmar, esférico y lateral).
- ✓ Realizar pruebas para verificar que los gestos de agarre se activen de forma correcta dependiendo de las señales EMG obtenidas.

1.5. Metodología de desarrollo

Para el desarrollo del presente proyecto se propone la siguiente metodología:

- ✓ Rediseño de los mecanismos de los dedos de la mano robótica tomada de referencia para facilitar su fabricación.
- ✓ Simulación de movimiento en SolidWorks® de los dedos de la mano para determinar las velocidades de los actuadores a utilizar.
- ✓ Selección de drivers, encoders y motorreductores de acuerdo al espacio disponible y velocidades calculadas.
- ✓ Modificación del diseño de la mano de 6 grados de libertad adaptando los nuevos motorreductores y encoders.
- ✓ Fabricación de las partes de la mano mediante una impresora 3D.
- ✓ Ensamblaje de mecanismos de la mano y montaje de motorreductores.
- ✓ Fabricación del socket y colocación de drivers.
- ✓ Establecimiento de comunicación entre el brazalete MYO™ y la Raspberry Pi 3® para la obtención de señales EMG preprocesadas.

- ✓ Implementación del clasificador de patrones de valor real basada en memoria asociativa extendida para catalogar actividades musculares similares.
- ✓ Comunicación entre la Raspberry Pi 3® y un microcontrolador ATmega para el control de motorreductores y control de posición.
- ✓ Simulación de gestos prensiles humanos en la mano y generación de base de datos de posiciones de cada motor que forman el gesto deseado.
- ✓ Relación de actividad muscular con gestos prensiles.
- ✓ Inclusión de los sistemas de control de posición, adquisición y procesamiento de datos a un solo sistema para ejecutarlos en conjunto.
- ✓ Realización de pruebas y obtención de resultados.

1.6. Estructura de la tesis.

El presente documento de tesis está organizado de la siguiente manera. En el primer capítulo se presenta una introducción y algunos antecedentes al tema planteado, se define el planteamiento y la justificación del problema, se especifican los objetivos a cumplir y por último se describe la metodología utilizada durante el desarrollo del trabajo. En el Capítulo 2 se proporcionan algunos conceptos teóricos utilizados en el desarrollo de la tesis, tales como la electromiografía, configuraciones de sujeción comunes de la mano humana y algunos mecanismos utilizados para el movimiento de la mano robótica. En el Capítulo 3 se presenta el desarrollo de rediseño, manufactura y ensamble del prototipo. El sistema de instrumentación del sistema protésico en el cual se describen los elementos electrónicos usados para medir y controlar el prototipo se aborda en el Capítulo 4. En el Capítulo 5 se presenta el modo de operación del sistema completo, mostrando la lógica que rige el comportamiento del prototipo y la manera en que ésta fue desarrollada. En el Capítulo 6 se presentan las pruebas realizadas para comprobar el funcionamiento de clasificación en el prototipo y verificar que el diseño permita sujetar diversos objetos mediante las estrategias de agarre. Por último, en el Capítulo 7 se exponen las conclusiones del trabajo desarrollado y las propuestas de trabajos futuros para dar seguimiento al desarrollo del prototipo.

Capítulo 2

Marco teórico

En este capítulo se presentan algunos conceptos y definiciones que serán de utilidad en el desarrollo de los sistemas que conformarán la prótesis electromiográfica, así como algunos proyectos que servirán de base para el desarrollo del prototipo y ayudarán a acelerar el proceso de diseño mecánico y de software.

2.1. Mecanismos utilizados para el movimiento de la mano robótica

Los mecanismos que utiliza la mano robótica para ejecutar los movimientos de los dedos están constituidos principalmente con engranes y bandas que transmiten el par generado por los motorreductores.

Como puede observarse en la Figura 2.1, los mecanismos utilizados en el diseño de cada dedo constan de trenes de engranes y una banda de sincronización.

Los trenes de engranes, conformados por engranes cónicos y rectos tienen la función de transmitir el par del motor y transformarlo en el movimiento de la articulación que une la palma con el dedo (flanges proximales)(ω_2). Estas a su vez, transmiten un movimiento mediante una banda de sincronización (ω_3) a la articulación del dedo (falange media)(ω_4).

Al mismo tiempo, en el movimiento de rotación del pulgar se utiliza un sistema que consiste en un tornillo sinfin y una corona sinfin (también llamada engrane de gusano) [20].

2.1.1. Tren de engranes

Un tren de engranes es un conjunto de dos o más engranes acoplados. Estos mecanismos servirán como transmisión de movimiento entre los actuadores y las falanges de los dedos de la mano robótica. Aquellos engranes que generan una fuerza sobre otro son llamados engranes motrices, mientras que a los que le son inducidos una fuerza externa son llamados elementos conducidos o impulsados. En un tren de engranes simple la velocidad del último elemento está definida por la ecuación (2.1.1) [21] [22].

$$n_L = en_F \tag{2.1.1}$$

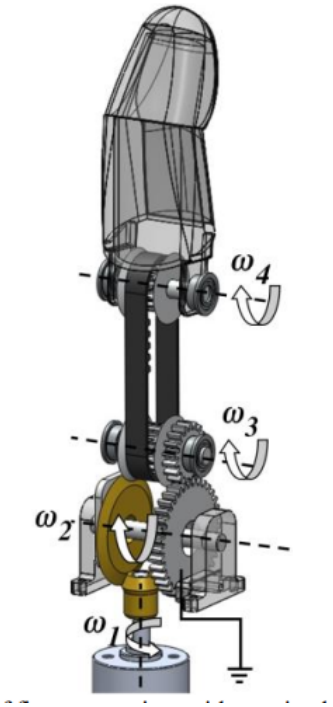


Figura 2.1: Mecanismo de movimiento de un dedo [20].

en donde n_L es la velocidad en rpm del engrane final, n_F es la velocidad en rpm del primer engrane y e es el *valor de tren* definido en la ecuación (2.1.2) [22].

$$e = \frac{producto\ del\ número\ de\ dientes\ de\ los\ motrices}{producto\ del\ número\ de\ dientes\ de\ los\ impulsados} \tag{2.1.2}$$

De esta forma, con las ecuaciones (2.1.1) y (2.1.2) se puede conocer la velocidad de cada falange de acuerdo a la velocidad de los motores.

2.1.2. Bandas de sincronización

Como puede observarse en la Figura 2.1, el mecanismo también cuenta con una banda que transmite el movimiento de una articulación a otra.

Las bandas de sincronización, como las utilizadas en el mecanismo de movimiento de los dedos, tienen dientes que entran en ranuras axiales formadas en las poleas por lo que no se estiran ni deslizan y transmiten potencia a una relación constante de velocidad angular. Están hechas de tela impregnada con caucho y con alambre de acero o algún otro material que ayude a resistir la carga de tensión. Este tipo de bandas pueden operar en un intervalo muy amplio de velocidades, no requieren lubricación y son más silenciosas que las transmisiones de cadena [22].

El elemento de tensión de una banda de sincronización se ubica en la línea de paso de la banda (Figura 2.2), de manera que la longitud de paso es la misma sin que importe el espesor

del respaldo.

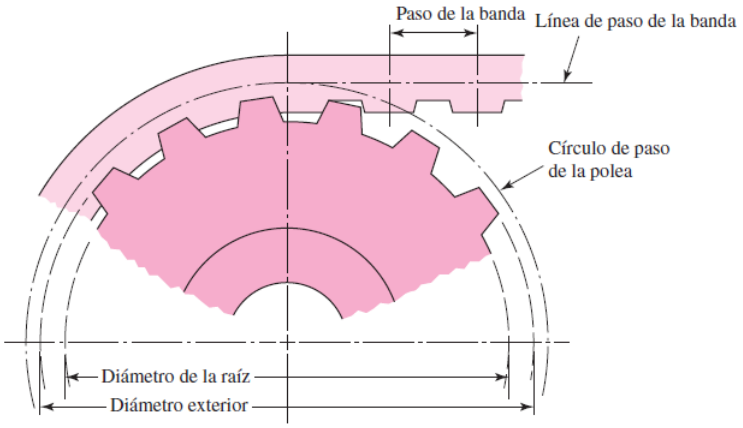


Figura 2.2: Partes de la polea y la banda de sincronización [22].

La longitud de paso Lp se determinan mediante la ecuación (2.1.3).

$$L_p = 2C + \frac{\pi(D+d)}{2} + \frac{(D-d)^2}{4C} \tag{2.1.3}$$

donde:
D = diámetro exterior de la polea mayor
d = diámetro exterior de la polea menor
C = distancia entre centros

De esta manera se podrán calcular las longitudes de las bandas de sicronización a implementar en los mecanismos del prototipo.

2.2. Configuraciones de sujeción comunes

Para el desarrollo del proyecto es necesario conocer y definir los principales tipos de agarre de la mano humana que se implementarán en el prototipo.

Es gracias a la gran cantidad de músculos y articulaciones que tiene la mano que es posible realizar una gran variedad de configuraciones de sujeción. Se ha hecho una clasificación de dichas configuraciones [13]:

Movimientos prensiles.
Son movimientos en los cuales un objeto es sostenido total o parcialmente dentro de la mano.

Movimientos no prensiles.
Son aquellos movimientos en los que no se realizan acciones de agarre, sin embargo los objetos pueden ser manipulados, empujados o trasladados con la mano entera o con los dedos.

Schlesinger, en 1919 propuso seis categorías en las cuales pretende describir las estrategias de agarre más comunes en la mano humana [13]. Dichas estrategias de agarre se implementarán en el prototipo para ampliar las actividades que el usuario pueda realizar e interactuar con un mayor número de objetos de manera más natural. Las categorías son las siguientes (ver Figura 2.3.):

- ✓ Agarre cilíndrico.
- ✓ Agarre de punta.
- ✓ Agarre tipo gancho.
- ✓ Agarre de palma (palmar).
- ✓ Agarre esférico.
- ✓ Agarre de lado (lateral).

Figura 2.3: Estrategias de agarre más comunes en la mano humana [23].

2.3. Electromiografía

El control del prototipo estará basado en señales electromigráficas. La electromiografía es el registro de la actividad eléctrica generada por el músculo liso o estriado ya sea de manera voluntaria o inconsciente [24]. Las señales electromiográficas (señales EMG) son señales eléctricas (de 5 a 20 µV) producidas por un músculo durante el proceso de contracción y relajación del mismo, proporcionando información acerca del funcionamiento de los músculos y nervios.

Las señales EMG representan una medida de la actividad muscular y pueden ser detectadas usando electrodos en la superficie de la piel (electromiografía superficial) o de manera interna mediante agujas insertadas en el músculo (electromiografía intramuscular), lo que permite controlar sistemas fisioeléctricos mediante la contracción de determinados músculos [6] [9].

El uso de la electromiografía determina el tipo de electrodo de registro, de manera que la electromiografía de aguja (intramuscular) se utiliza generalmente para distinguir entre lesiones del sistema nervioso central (SNC) y el sistema nervioso periférico (SNP), mientras que la electromiografía de superficie (EMGS) tiene aplicaciones enfocadas mayormente a la rehabilitación [25].

Las señales EMG obtenidas mediante electromiografía de superficie son captadas a través de electrodos que entran en contacto con la piel en la zona donde se encuentra el músculo que generará la señal. Una vez captada, se amplifica y se procesa en un controlador que se encarga de llevar a cabo la funcionalidad del sistema a controlar [6] [5].

2.4. Clasificadores

La clasificación consiste en asignar un dato, objeto o instancia a una categoría o clase existente. Básicamente ésto permite abstraer la información, llevándola a una representación más adecuada para la toma de decisiones.

En el desarrollo de la prótesis mioeléctrica se implementará un clasificador para catalogar las señales EMG obtenidas con el brazalete MYO y así activar la estrategia de agarre deseada en el prototipo cuando la actividad muscular sea la indicada para dicho gesto.

El proceso de clasificación consiste, desde el punto de vista matemático, en asignar una instancia representada por un vector de características o atributos $X = X1, X2, ..., Xm$ a una clase c de un conjunto de clases C [26].

Existen dos tipos de clasificadores:

No supervisado o agrupamiento.
En este tipo de clasificadores las clases son desconocidas, y el problema consiste en dividir un conjunto de n objetos en k clases, de forma que a objetos similares se les asigne una misma clase.

Supervisado.
Las clases se conocen desde un inicio y el problema consiste en encontrar una función que asigne a cada objeto su clase correspondiente.

2.4.1. Memorias Asociativas Extendidas

En 2004, H. Sossa et al. presentaron un modelo de memorias asociativas capaz de clasificar patrones con elementos de valor real [27]. Este modelo, llamado Memorias Asociativas Extendidas (MAE), es una extensión del conocido modelo propuesto por Steinbuch, la Learmatrix [28], el cual opera únicamente con patrones cuyos elementos tienen valor binario. El modelo MAE está basado en un concepto general de la función de aprendizaje de una memoria

asociativa. Este modelo ha demostrado un alto desempeño en la clasificación de patrones que tienen componentes de valor real, así como en versiones alteradas de los patrones utilizados en la fase de entrenamiento [29].

Fase de entrenamiento de una MAE

Antes de describir la fase de entrenamiento de este modelo, es necesario introducir al elemento que permite relacionar patrones con las clases a las que ellos pertenecen.

Definición 1. Sea $(c_1, c_2, ...c_i, ..., c_N)$ un conjunto de N clases, donde cada clase agrupa un cierto número de patrones, estos elementos forman al conjunto fundamental de asociaciones. Entonces existe una función, llamada ϕ, cuyo objetivo es codificar la información de todos los patrones que pertenecen a la clase i y ubicarla en la $i-$*ésima* fila de la memoria asociativa, definida por la matriz $\mathbf{M}$. Ahora, la fase de entrenamiento consiste en evaluar la función ϕ para cada clase. Por lo que la matriz $\mathbf{M}$ puede ser estructurada de la siguiente forma:

$$\mathbf{M} = \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_N \end{bmatrix} \tag{2.4.1}$$

donde ϕ_i es la evaluación de ϕ para todos los patrones de la clase i, siendo $i = 1, 2, ..., N$. La función ϕ puede ser evaluada de diferentes formas. El operador promedio aritmético (**prom**) es frecuentemente usado en tratamiento de señales e imágenes. En [27], los autores estudiaron el desempeño del operador **prom** y del operador mediana (**med**) en la evaluación de la función ϕ. Finalmente, el objetivo de la fase de entrenamiento es establecer la relación existente entre un vector de entrada $\mathbf{x} = [x_j]_n$, y la clase c_i a la que pertenece. Considerando que cada clase está integrada por q patrones $\mathbf{x} = [x_j]_n$, y que $\phi_i = (\phi_{i,1}, ..., \phi_{i,n})$. Entonces, la fase de entrenamiento de la MAE, cuando el operador **prom** es usado para evaluar la función ϕ_i, es definida por:

$$\phi_{i,j} = \frac{1}{q} \sum_{l=1}^{q} x_{j,l}, j = 1, ..., n \tag{2.4.2}$$

Por otro lado, la fase de entrenamiento de la MAE, cuando el operador **med** es usado para evaluar la función ϕ_i, es definida por:

$$\phi_{i,j} = \underset{l=1}{\overset{q}{\mathbf{med}}}\ x_{j,l}, j = 1, ..., n \tag{2.4.3}$$

La memoria asociativa $\mathbf{M}$ es obtenida después de evaluar todas las funciones ϕ_i. Para el caso donde existen N clases y los vectores a clasificar son $n - dimensionales$, la memoria resultante $\mathbf{M} = [m_{ij}]_{N \times n}$ es:

$$\mathbf{M} = \begin{bmatrix} \phi_{1,1} & \phi_{1,2} & \dots & \phi_{1,n} \\ \phi_{2,1} & \phi_{2,2} & \dots & \phi_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{N,1} & \phi_{N,2} & \dots & \phi_{N,n} \end{bmatrix} = \begin{bmatrix} \mathbf{m}_{1,1} & \mathbf{m}_{1,2} & \dots & \mathbf{m}_{1,n} \\ \mathbf{m}_{2,1} & \mathbf{m}_{2,2} & \dots & \mathbf{m}_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{m}_{N,1} & \mathbf{m}_{N,2} & \dots & \mathbf{m}_{N,n} \end{bmatrix} \tag{2.4.4}$$

Fase de clasificación de una MAE

El objetivo de la fase de clasificación es generar el índice de la clase a la que un vector de entrada pertenece. El proceso de clasificación realizado por la MAE da inicio cuando un patrón $\mathbf{x}^{\mu} \in \mathbf{R}^{n}$ es presentado a la memoria $\mathbf{M}$ que fue generada en la fase de entrenamiento. Una MAE es capaz de clasificar un patrón que no necesariamente haya sido usado para construir la matriz $\mathbf{M}$, incluyendo versiones alteradas por ruido de vectores usados en dicha fase. Cuando el operador **prom** es usado en la fase de entrenamiento, la clase a la cual pertenece el vector de entrada $\mathbf{x}$ es definida por:

$$i = \underset{l}{arg} \left[\overset{N}{\underset{l=1}{\wedge}} \overset{n}{\underset{j=1}{\vee}} |m_{lj} - x_j| \right] \tag{2.4.5}$$

En este caso, los operadores $\wedge \equiv MIN$ y $\vee \equiv MAX$ desarrollan operaciones morfológicas sobre valores absolutos de las diferencias de los componentes m_{ij} de $\mathbf{M}$ y los componentes x_j del patrón $\mathbf{x}$ que será clasificado. Cuando el operador **med** es usado en la fase de entrenamiento, la clase a la cual pertenece el vector de entrada $\mathbf{x}$ es definida por:

$$i = \underset{l}{arg} \left[\overset{N}{\underset{l=1}{\wedge}} \left| \underset{j=1}{\overset{n}{\mathbf{med}}} \, m_{lj} - \underset{j=1}{\overset{n}{\mathbf{med}}} \, x_j \right| \right] \tag{2.4.6}$$

En este caso, el operador $\wedge \equiv MIN$ desarrolla una operación morfológica de erosión sobre valores absolutos de las diferencias de las medianas de los componentes m_{ij} de $\mathbf{M}$ y los componentes x_j del patrón $\mathbf{x}$ que será clasificado. El Teorema 1 y el Corolario 1-5 definidos en [27] gobiernan las condiciones que deben ser cumplidas para obtener un proceso de clasificación perfecto.

2.5. Proyecto myo-raw

Dzhu, usuario de GitHub (plataforma de desarrollo en donde se pueden alojar y revisar códigos, administrar proyectos y crear software en su mayoría de código abierto) creó una interfaz de comunicación con el Brazalete MYO™ de Thalmic™ (brazalete de obtención de datos electromiográficos), permitiendo la conexión con un brazalete cercano, dándole acceso a información de los sensores EMG y de un reconocimiento de gestos precargado.

El objetivo principal de este proyecto es generar una interfaz rápida que permita la obtención de datos de los sensores EMG y de esta manera facilitar investigaciones donde se usen dichos datos en algoritmos de control que permitan aplicar esta tecnología en prótesis u otros campos como la robótica [30].

El script de Python, liberado en el 2014 es actualizado constantemente y contiene las implementaciones necesarias del protocolo Bluetooth para realizar la conexión y configuración del brazalete MYO™. Al ejecutarse, automáticamente muestra una gráfica de los valores EMG preprocesados por el brazalete en tiempo real.

Este script se implementará en la prótesis mioeléctrica para realizar la conexión entre el brazalete y la Raspberry Pi 3® encargada de procesar las señales.

Capítulo 3

Desarrollo de la mano robótica

Se opto por el uso del diseño del prototipo propuesto en [20], ya que la mano robótica puede realizar diversos gestos de agarre y facilitar la manipulación de objetos, además de resultar económico fabricar la mayoría de partes de la mano mediante una impresora 3D.

El prototipo originalmente cuenta con 6 grados de libertad y 10 juntas móviles, utilizando un motor de CD en cada dedo y un motor para rotar el pulgar haciéndolo opuesto a los otros 4.

El mecanismo utilizado para realizar los movimientos de los 4 dedos opuestos al pulgar se muestra en la Figura 2.1, mientras que en la Figura 3.1 se observa el diseño del mecanismo original utilizado para ejecutar los movimientos en el dedo pulgar.

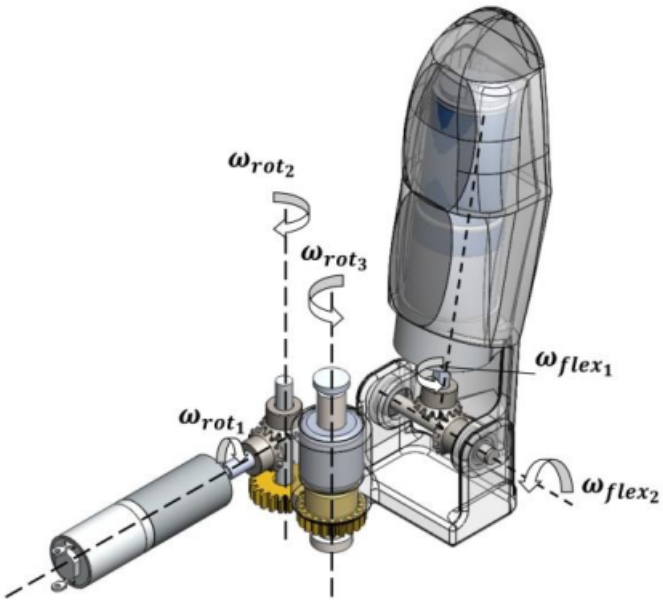


Figura 3.1: Mecanismo de movimiento del dedo pulgar [20].

A pesar de que la mayor parte de la mano puede ser impresa en 3D, los mecanismos utilizados para la transmisión de movimiento son de metal y son costosos (según [20] el costo de todos los engranes rondaba los $\$1,500$ dólares en 2013). Dado este contexto, se buscó

la manera de reducir costos para su fabricación utilizando engranes de otro material. De la misma manera, se decidió cambiar los motores de CD propuestos en el artículo original por los micromotores de metal con eje extendido (HPCB) de la marca Pololu® que presenta una relación de reducción de 150:1, una velocidad de 200 RPM (sin carga, 6V) y una corriente a motor parado de 1.5A [31].

Los controladores utilizados para manipular la dirección de giro y la velocidad de los motores son los Pololu Dual MC33926 Motor Driver [32]. Dichos dispositivos cuentan con una corriente de salida continua de hasta 3A por motor, compatible con sistemas electrónicos de 3.3V o 5V y control de velocidad de motor mediante PWM (modulación por ancho de pulsos).

Para obtener la posición de los motores se eligieron los encoders magnéticos de la misma marca compatibles con los micromotores a utilizar. Sus especificaciones se pueden consultar en [33].

Debido a estos cambios, fue necesario modificar el diseño original con los nuevos engranes tomando en cuenta el nuevo tamaño de los motores y encoders, además de añadir un movimiento dependiente en el dedo pulgar (entre la falange distal y proximal) para mejorar su versatilidad, aumentando las juntas móviles de la mano a 11.

3.1. Modificaciones al diseño original

En la Figura 3.2 se puede observar el diseño modificado con los nuevos engranes utilizados para la transmisión de movimiento de los dedos opuestos al pulgar.

El movimiento de los dedos ocurre cuando el engrane 4 comienza a girar. Esto provoca que el engrane 5 (unido a la falange proximal del dedo) gire sobre su eje (perpendicular al eje del micromotor) llevándose a cabo el movimiento de la falange proximal. Al realizarse tal movimiento, el engrane 2 girará alrededor del engrane 3 (el cual está fijo a la palma de la mano) y generará un movimiento rotatorio sobre su eje. Al contener una polea dentada unida a sí mismo, el movimiento generado en el engrane 2 se transmitirá a través de la banda dentada 6 y se reflejará en la polea dentada unida al engrane 1. Dicho engrane al estar acoplado a la falange distal del dedo, propiciará su movimiento al mismo tiempo que la falange proximal.

Los engranes 1, 2 y 3 tienen el mismo número de dientes y el mismo diámetro. De la misma manera, las poleas unidas a los engranes 1 y 2 tienen las mismas dimensiones entre sí; gracias a esto y a que la banda es dentada, el movimiento angular que recorre el engrane 1 será el mismo en el engrane 2. El ángulo entre la palma y las falanges de los dedos dependerá proporcionalmente de la posición del eje de salida del motorreductor a utilizar. La relación de los engranes 4 y 5 es 4.5:1, lo que indica que el eje del motor deberá dar 4 vueltas y media (1620°) para que el engrane 4 gire 1 vuelta completa (360°).

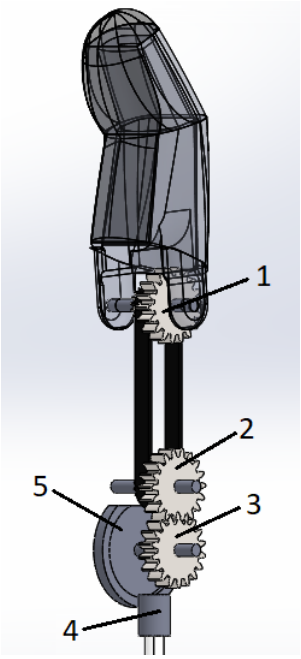


Figura 3.2: Mecanismo de movimiento de los dedos opuestos al pulgar.

El mecanismo de rotación del pulgar fue sustituido y se agregó una banda de sincronización para transmitir el movimiento al eje añadido en el pulgar. Ambos diseños se pueden observar en la Figura 3.3.

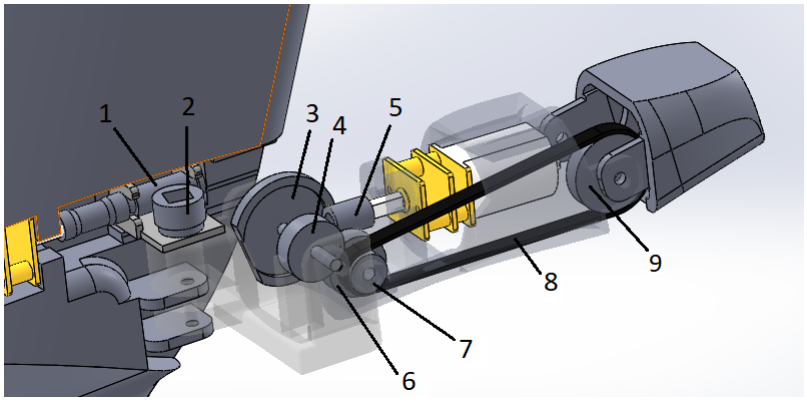


Figura 3.3: Mecanismo de movimiento de rotación y flexión del dedo pulgar.

El mecanismo de rotación original fue reemplazado con el fin de obtener el mismo movimiento con menor cantidad de engranes y de diferentes dimensiones, en este caso utilizando un tornillo sinfin y una corona sinfín (elementos 1 y 2 respectivamente de la Figura 3.3). La relación entre los engranes de rotación de pulgar es 14:1, lo que indica que el eje de salida del motorreductor deberá girar 14 vueltas (de 360°) para mover el engrane del pulgar 360°. El movimiento de rotación se genera al activarse el micromotor acoplado al tornillo sinfín, el cual hace rotar al elemento 2 acoplado a todo el dedo pulgar. El movimiento de flexión del dedo pulgar se genera cuando el engrane 5 comienza a girar, provocando que el motor junto

con la falange proximal del pulgar giren alrededor del engrane 3 fijo a la base del dedo. De manera similar al de los otros dedos, el engrane unido a la polea dentada (7) rodea al engrane 6 también fijo a la base del dedo. Dicho movimiento se traduce en un giro del engrane 7, transmitiendo el movimiento por medio de la polea dentada 8 a la polea 9 unida a la falange proximal. Los engranes 3 y 5 (Figura 3.3) determinan el ángulo entre la base del pulgar y el dedo. Su relación es igual a la implementada para los demás dedos (4.5:1). Los elementos 4 y 6 son iguales por lo que su movimiento será de igual magnitud cambiando únicamente el sentido de giro. La polea identificada con el número 7 se mueve en conjunto con el engrane 6. Respecto a los elementos 9 y 7 tienen una relación 2:1 y están en contacto con la banda de sincronización 8, por lo que el elemento 7 al dar una vuelta (360°) la falange distal del pulgar se moverá 180° respecto al dedo.

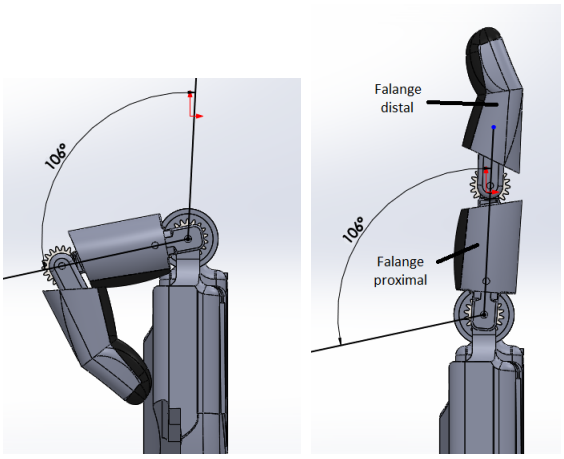
Con estos datos y las ecuaciones (2.1.1) y (2.1.2) se calcularon los tiempos máximos que tardará cada dedo en ir de su posición inicial (dedos completamente cerrados) a su posición final (dedos en una posición de la mano abierta) considerando la velocidad máxima de los micromotores proporcionada por el fabricante (200 RPM).

Para los movimientos de apertura y cerradura de los dedos se tiene un valor de tren $e = \frac{1}{4{,}5}$ tomando en cuenta únicamente los engranes 4 y 5 de la Figura 3.2, al ser los encargados de realizar el movimiento independiente. De este modo la velocidad angular será:

$$n_L = (\frac{1}{4{,}5})(200RPM) = 44{,}445RPM \tag{3.1.1}$$

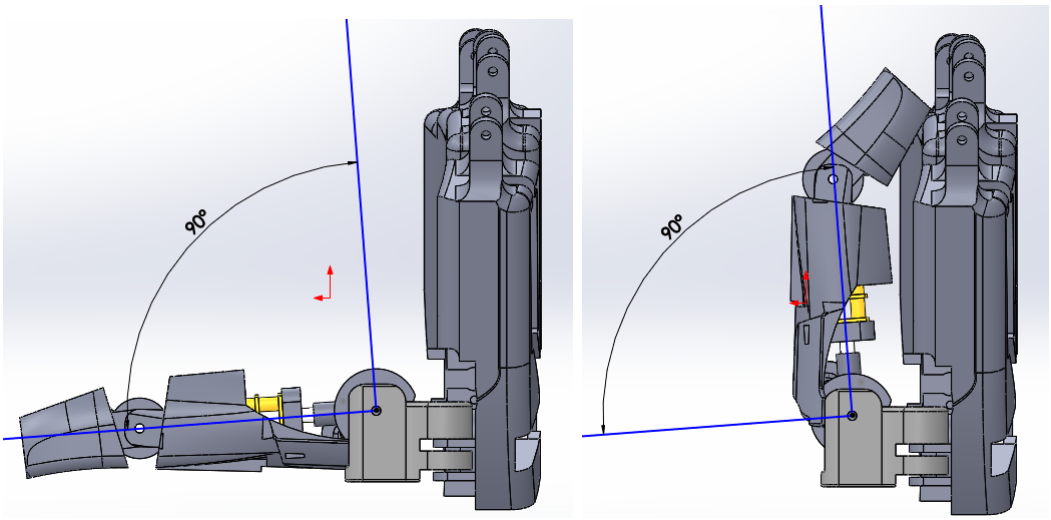
Realizando las conversiones requeridas se obtiene que para girar 1° transcurrirán 0.0037 segundos.

Estimando un rango de movimiento de 106° para los dedos opuestos al pulgar y de 90° para el pulgar (Figuras 3.4 y 3.5), se calcula que el tiempo aproximado en realizar dicho movimiento de apertura o cerradura será de 0.392 segundos por cada dedo opuesto al pulgar y de 0.333 segundos para el pulgar.



(a) (b)

Figura 3.4: Ángulo de movimiento de los dedos opuestos al pulgar.(a)Dedo en su posición cerrada. (b)Dedo en su posición abierta.



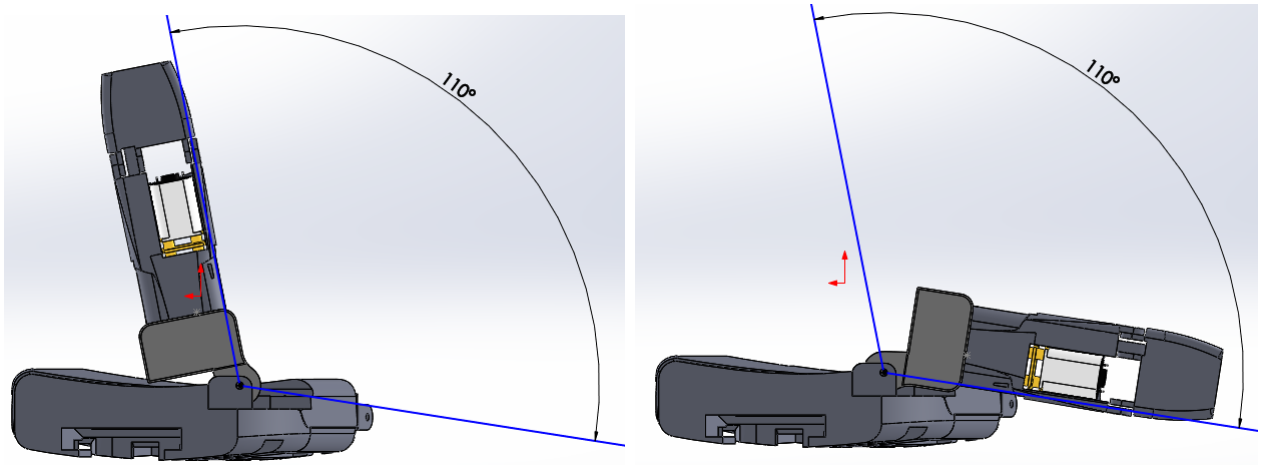
(a) (b)

Figura 3.5: Ángulo de movimiento del pulgar.(a)Posición cerrada. (b)Posición abierta.

De la misma manera se realizan los cálculos para el movimiento de rotación del pulgar. Considerando un valor de tren $e = \frac{1}{14}$ se tiene:

$$n_L = (\frac{1}{14})(200RPM) = 14{,}285RPM \tag{3.1.2}$$

Una vez más, al realizar las conversiones necesarias se valora que el movimiento de 1° de rotación del pulgar tardará 0.0116 segundos, dando un tiempo total de rotación de 1.276 segundos considerando un rango de movimiento de 110° (Figura 3.6).



(a) (b)

Figura 3.6: Ángulo de movimiento de rotación del pulgar.(a)Posición cerrada. (b)Posición abierta.

Una vez realizados los cálculos de las velocidades, se procedió a simular los movimientos en SolidWorks® para observar el comportamiento de los dedos con los mismos parámetros de velocidad (200RPM) y de rango de movimiento. En la Figura 3.7 se puede observar la gráfica de la velocidad angular del micromotor contra el tiempo transcurrido mientras se realiza un movimiento de apertura (representada desde el tiempo 0 hasta que la velocidad se vuelve 0) y cerradura del dedo (representada en la gráfica desde que la velocidad aumenta nuevamente partiendo de 0 hasta que la gráfica termina). Se puede apreciar que el tiempo que transcurre mientras el dedo se cierra completamente (Figura 3.4a) desde una posición abierta (ver Figura 3.4b) es de aproximadamente 0.4 segundos.

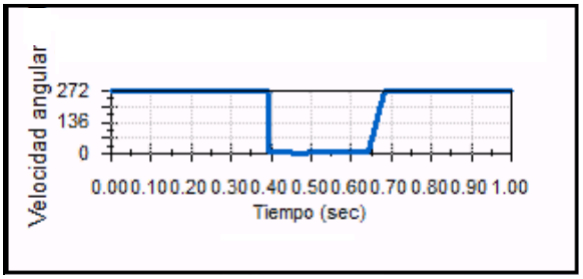


Figura 3.7: Gráfica de la velocidad del motor para abrir y cerrar un dedo opuesto al pulgar.

De igual manera, se obtuvieron las gráficas para los movimientos de flexión del pulgar para abrir y cerrar el dedo y para el movimiento de rotación del mismo, contemplando únicamente la cerradura del dedo con la palma. Dichas gráficas se observan en las Figuras 3.8 y 3.9 respectivamente.

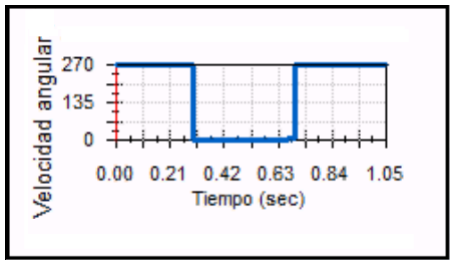


Figura 3.8: Gráfica de la velocidad del motor para abrir y cerrar el dedo pulgar.

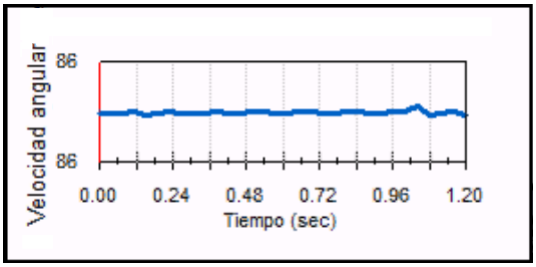


Figura 3.9: Gráfica de la velocidad del motor para rotar el dedo pulgar.

La gráfica de la Figura 3.8 indica que el movimiento de flexión del pulgar se realizará en aproximadamente 0.3 segundos si el motor gira con una velocidad de 200 RPM. El tiempo de rotación del mismo dedo es aproximadamente 1.2 segundos (Figura 3.9).

Con base en los resultados obtenidos y una vez verificados los cálculos en la simulación, se hizo una aproximación del tiempo máximo que tardará el sistema para realizar un gesto, suponiendo que para realizarlo se tendrán que activar todos los motores uno a uno.

De esta manera se puede predecir que la realización de un gesto tardará aproximadamente 3.1 segundos (1.6 segundos para los dedos opuestos al pulgar, 1.2 segundos para la rotación del pulgar y .3 segundos para la flexión del mismo).

Se debe tomar en cuenta que el tiempo máximo para generar un gesto podrá ser mayor dependiendo de la velocidad con la que se ejecuten los movimientos, ya que las simulaciones se realizaron con la velocidad máxima de los motores.

En el Apéndice A se muestran las especificaciones de cada una de las piezas de la mano robótica, seguida de ensambles y vistas explosionadas del prototipo. En la Tabla 3.1 se presenta una lista de todos los elementos que la conforman.

No. de dibujo	Título	Cantidad
FDO	Falanges distales de los dedos opuestos al pulgar	4
FPO	Falanges proximales de los dedos opuestos al pulgar	4
FDP	Falange distal del dedo pulgar	1
FPP	Falange proximal del dedo pulgar	1
BDP	Base del dedo pulgar	1
PM	Palma de la mano	1
STS	Soporte para tornillo sinfín	1
E18	Engrane de 18 dientes	4
EP1818	Engrane con polea de 18-18 dientes	8
TSF	Tornillo sinfín	1
P40	Polea de 40 dientes	1
E17	Engrane de 17 dientes	1
EP1718	Engrane con polea de 17-18 dientes	1
C14	Corona con 14 dientes	1
EA3	Eje de articulación de 3.5 mm	1
EA8	Eje de articulación de 8.5 mm	1
EA11	Eje de articulación de 11 mm	1
EA18	Eje de articulación de 18 mm	12
EA27	Eje de articulación de 27 mm	1
E8	Engrane de 8 dientes	5
BD	Banda dentada	4
BDP	Banda dentada del dedo pulgar	1
EP	Engrane perpendicular	5
MM	Micromotor	6

Tabla 3.1: *Lista de elementos de la mano robótica*

Finalmente, en la Figura 3.10 se pueden observar las dimensiones aproximadas de la mano ensamblada.

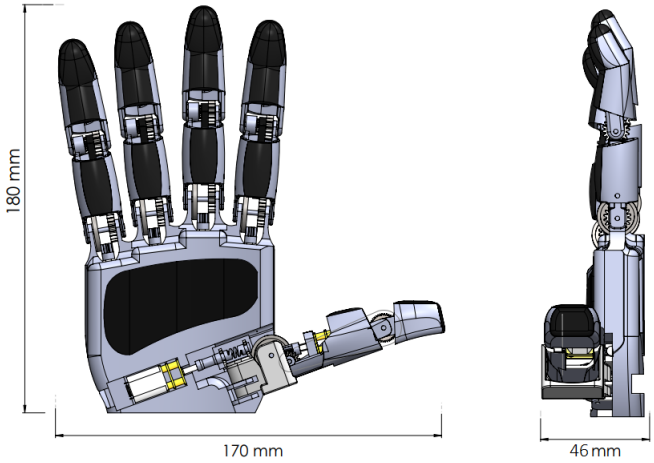


Figura 3.10: Dimensiones de la mano ensamblada.

3.2. Fabricación y ensamblaje de la mano robótica.

La carcasa de la mano fue diseñada para imprimirla en 3D, por lo que las piezas que conforman los dedos y la palma de la mano se fabricaron en una impresora 3D de la marca Mbot, el material utilizado es PLA de diámetro 1.75mm. Las piezas impresas se pueden observar en la Figura 3.11.

Figura 3.11: Partes de la mano impresas en 3D.

Una vez impresas, las piezas se lijaron para quitar las imperfecciones y se colocaron fragmentos de caucho en las partes internas de la mano para mejorar el agarre de objetos. Tomando en cuenta la distancia entre centros de los engranes 2 y 3 de la Figura 3.2 y el

diámetro de las poleas dentadas a utilizar para la transmisión de movimiento con la banda de sincronización, se aplicó la ecuación (2.1.3) para obtener la longitud de las bandas a utilizar:

Para los dedos opuestos al pulgar:

$$L_p = 2(38{,}5mm) + \frac{\pi((6mm) + (6mm))}{2} + \frac{((6mm) - (6mm))^2}{4(38{,}5mm)} = 95{,}84mm \qquad (3.2.1)$$

siendo:
6 mm el diámetro de ambas poleas (mayor y menor) y 38.5mm la distancia entre centros.

Para el pulgar:

$$L_p = 2(51{,}5mm) + \frac{\pi((12mm) + (6mm))}{2} + \frac{((12mm) - (6mm))^2}{4(51{,}5mm)} = 131{,}44mm \qquad (3.2.2)$$

siendo:
12 mm el diámetro de la polea mayor, 6 mm el diámetro de la polea menor y 51.5mm la distancia entre centros.

Una de las bandas y los engranes utilizados pueden observarse en la Figura 3.12. Una vez obtenidas todas las piezas a utilizar se procedió a ensamblar la mano robótica.

Figura 3.12: Banda y engranes utilizados para los mecanismos de la mano robótica.

Inicialmente se ensamblaron los dedos con sus mecanismos correspondientes como se muestra en las Figuras 3.13 y 3.14.

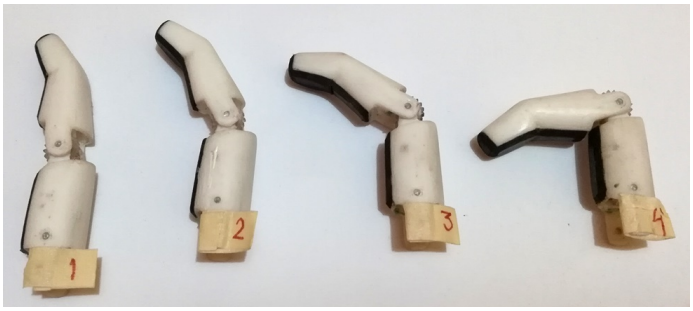


Figura 3.13: Dedos opuestos al pulgar ensamblados.

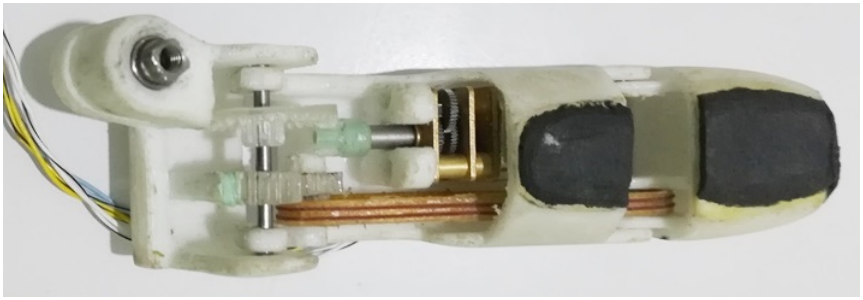


Figura 3.14: Dedo pulgar ensamblado.

En la Figura 3.15 se muestra el montaje de los encoders magnéticos a los motorreductores. Se soldaron los cables correspondientes a un conector para facilitar el montaje y desmontaje de la mano (ver Figura 3.16).

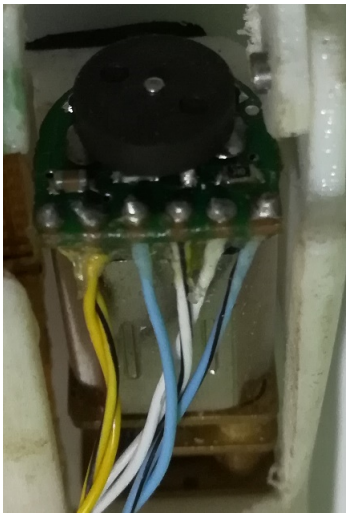


Figura 3.15: Encoder magnético soldado al motorreductor.

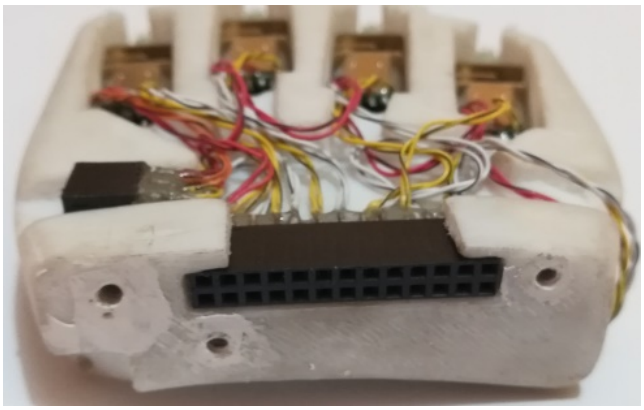


Figura 3.16: Conector soldado a los encoders.

Una vez acoplados los encoders y cables, se ubicaron y atornillaron los motorreductores en las piezas impresas (Figuras 3.17, 3.18 y 3.19). Además, se ensamblaron los engranes faltantes a los ejes de los motores. Una vez situados todos los elementos se colocaron los dedos para formar la mano completa.

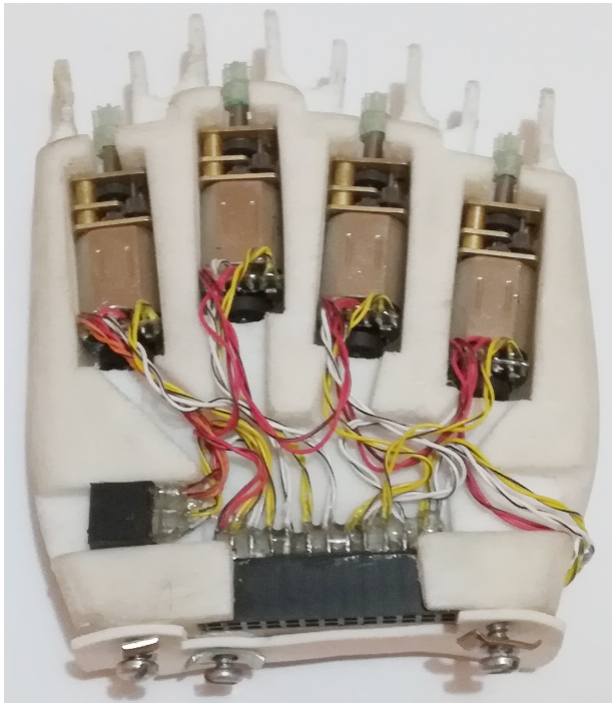


Figura 3.17: Motores colocados en el dorso de la mano.

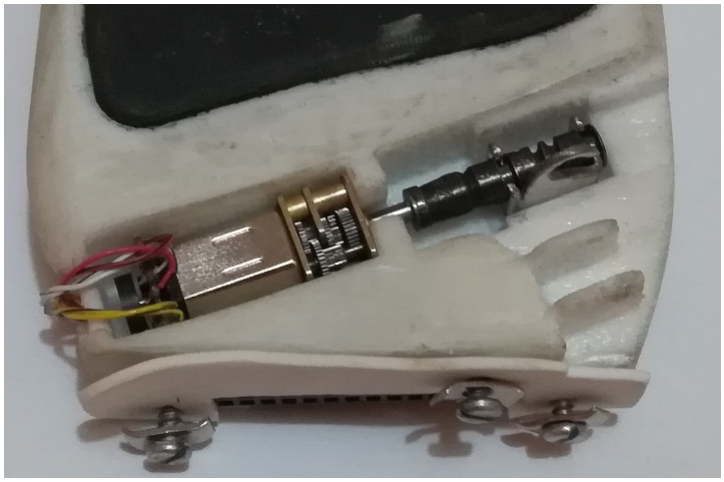


Figura 3.18: Motor de la palma de la mano.

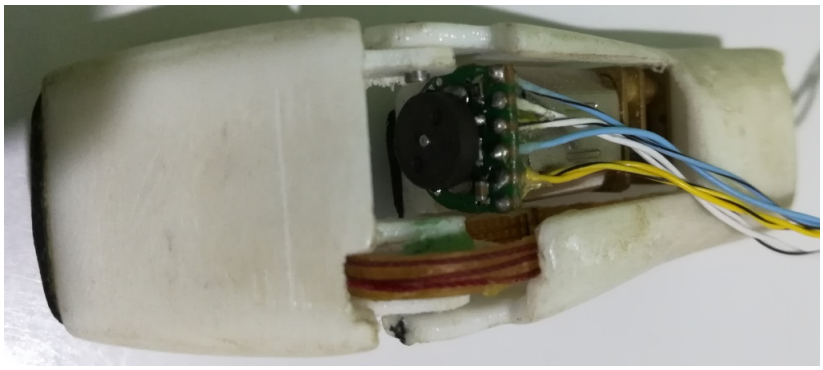


Figura 3.19: Motor del dedo pulgar.

La Tabla 3.2 contiene los costos de los elementos utilizados en el prototipo y los precios proporcionados en [20] para los elementos utilizados en el sistema original y que fueron sustituidos. Nótese que los precios mostrados en la referencia son del año 2013 y han sido convertidos de dólares estadounidenses a pesos mexicanos (considerando 1 dólar = $ 19.65 pesos, costo del dólar en diciembre del año 2018) para facilitar la comparación, por lo que los valores mostrados en la tabla deben ser considerados como una aproximación.

Componente	Costo original	Costo prototipo
Motores y reductores	$10200,31	$2766
Engranes, bandas y poleas	$29726,52	$850
Total	$39926,83	$3616

Tabla 3.2: *Comparación de costos de los componentes sustituidos.*

3.3. Fabricación del socket y colocación de drivers.

Para la elaboración del socket primero se fabricó un molde de yeso y posteriormente se recubrió con fibra de vidrio. Tras esperar a que secara y la colocación de varias capas de fibra de vidrio, se hicieron las ranuras necesarias para que los cables utilizados atravesaran el socket y se completara la comunicación de los drivers y los motorreductores/encoders. Los drivers se fijaron a la parte externa del socket mediante tornillos para dejar espacio en la parte interna del socket para introducir un brazo amputado (Figura 3.20).

Figura 3.20: Socket con drivers y cables colocados.

El ensamblaje de la mano robótica y el socket se muestra en la Figura 3.21.

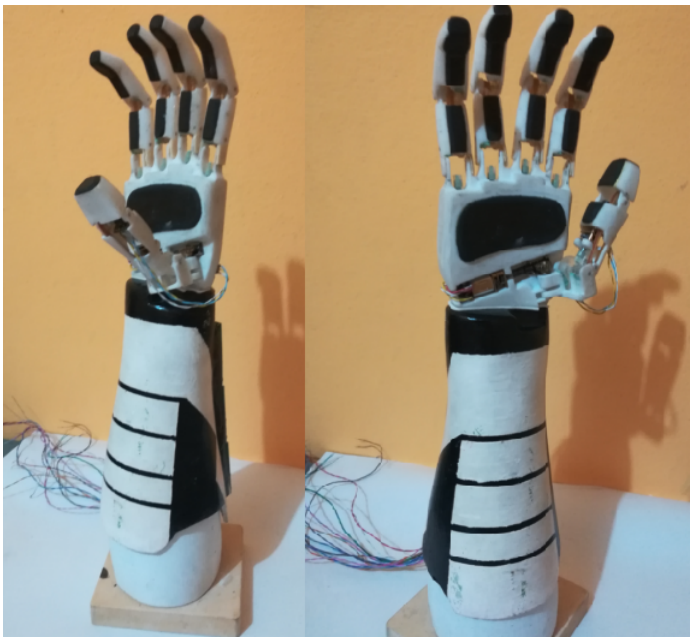


Figura 3.21: Prótesis ensamblada.

Capítulo 4

Instrumentación del prototipo

El sistema de instrumentación del prototipo tiene como función principal detectar la posición de los dedos de la mano robótica y sensar la actividad muscular en el brazo del usuario, además de realizar una comunicación entre los elementos de sensado, un ordenador de placa reducida y un microcontrolador para el procesamiento de señales y control de los actuadores.

4.1. Sensores de posición

Los sensores de posición utilizados son los encoders magnéticos Pololu enc03b (Figura 4.1), los cuales están acoplados al eje extendido del micromotor.

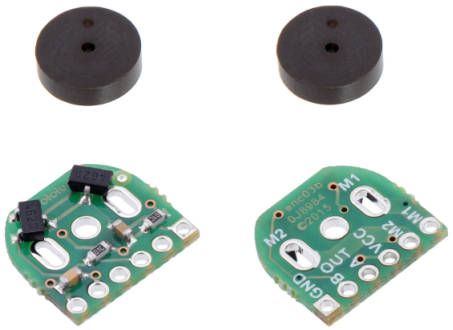


Figura 4.1: Encoder magnético Pololu modelo enc03b [33].

Dichos dispositivos cuentan con dos entradas de alimentación (2.7 V - 18 V) y dos canales de salida (A y B) que proporcionan trenes de pulsos con 6 cambios de estado cada uno por revolución. Los 12 cambios de estado generados sirven para conocer la posición y el sentido de giro del micromotor gracias a un desfase entre las dos señales (Figura 4.2); sin embargo se utilizó únicamente una salida de cada encoder para reducir el número de pines a utilizar y facilitar el procesamiento de datos. Debido a esto los cambios de estado provistos por el encoder se redujeron a 6 por cada revolución que el eje extendido del micromotor realiza.

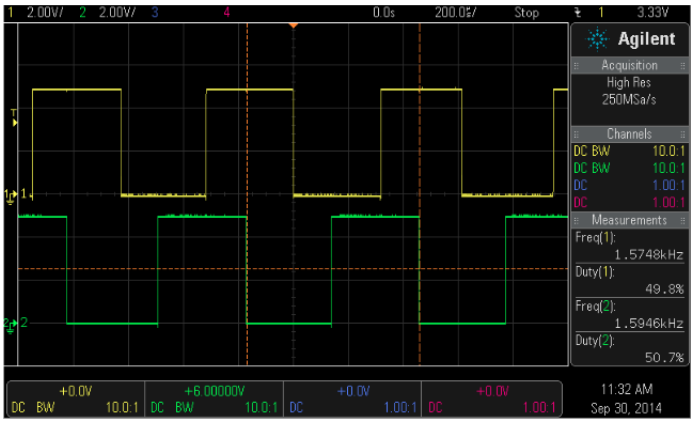


Figura 4.2: Tren de pulsos generado por los canales A y B del encoder magnético [33].

De esta manera y tomando en cuenta que las revoluciones sensadas son en el eje del motor y no a la salida del motorreductor, se procede a calcular el número de cambios de estado que generan los encoders al llevarse a cabo una revolución en el eje de salida del motorreductor.

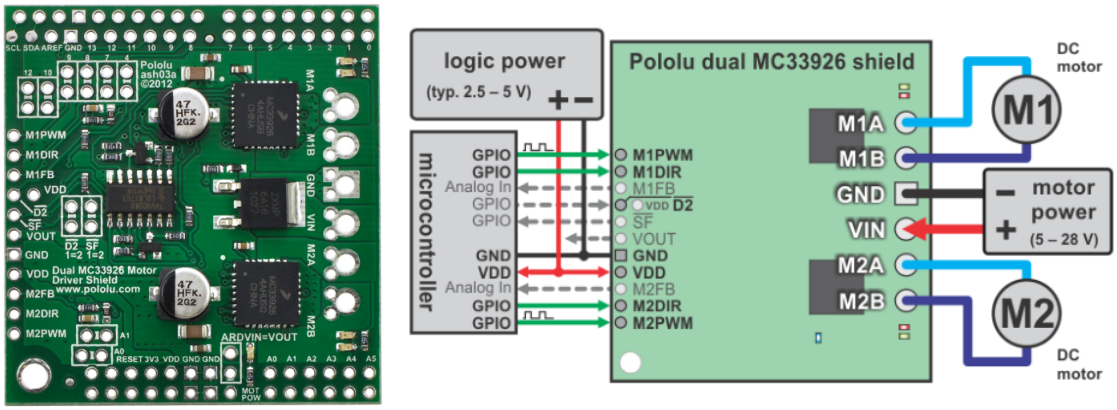
Al tener una relación 150:1, el eje del micromotor efectuará 150 revoluciones por cada giro del eje de salida. Así se generarán 900 cambios de estado en el encoder por cada giro del motorreductor. Debido a que la relación de los engranes encargados del movimiento de la falange proximal de los dedos con respecto a la palma de la mano (engranes 4 y 5 de la Figura 3.2) es 4.5:1 se generarán 4050 flancos por cada vuelta completa que realice el dedo. En la Tabla 4.1 se muestra el número de flancos generados por el encoder magnético con los ángulos que girará cada engrane.

Movimiento de abrir y cerrar dedos				
Eje de motor	Eje final de motorreductor	Engrane 4 (Fig. 3.2)	Engrane 5 (Fig. 3.2)	Flancos generados por el encoder
360°	2,40°	2,40°	0,533°	6
54000°	360°	360°	80°	900
243000°	1620°	1620°	360°	4050
Movimiento de rotación del dedo pulgar				
Eje de motor	Eje final de motorreductor	Engrane 1 (Fig. 3.3)	Engrane 2 (Fig. 3.3)	Flancos generados por el encoder
360°	2,40°	2,40°	0,171°	6
54000°	360°	360°	25,74°	900
756000°	5040°	5040°	360°	12600

Tabla 4.1: *Relación entre ángulos y flancos generados por el encoder*

4.2. Controlador de motores

Para controlar la velocidad y el sentido de giro de los micromotores, se utilizaron 3 controladores duales de motores de CD mostrados en la Figura 4.3a. Dichos controladores operan los motores con un voltaje de 5 a 28 V con una corriente de 3 A por canal y se alimentan con un voltaje de 2.5 a 5 V. La conexión con un microcontrolador se puede observar en la Figura 4.3b.



(a) (b)

Figura 4.3: (a)Controlador Dual MC33926 Pololu. (b)Esquema de conexión del módulo controlador con un microcontrolador.

A continuación se describe el funcionamiento de los pines a utilizar:

Los pines M1PWM y M2PWM son las entradas de las señales PWM que determinan la velocidad de giro de cada motor (1 y 2). Una señal con un ciclo de trabajo de 100 % hará que el motor correspondiente gire a su máxima velocidad, mientras que con un ciclo de trabajo de 0 % el motor quedará estático.

Los pines M1DIR y M2DIR son las entradas lógicas que indicarán el sentido de giro de cada motor. Si su valor lógico es 1 (alto) el motor correspondiente girará en un sentido. De manera contraria, si su valor lógico es 0 (bajo) girará en el otro sentido.

Los pines M1A, M1B, M2A y M2B son las salidas que se conectan a cada motor (M1A y M1B para un motor y M2A y M2B para otro).

Los pines VDD y GND son las conexiones para la alimentación de la parte lógica (2.5 a 5V) y tierra respectivamente.

Los pines VIN y GND son las conexiones para la alimentación de los motores (5 a 28 V) y tierra respectivamente.

4.3. Sensor electromiográfico

El brazalete electromiográfico MYO™ desarrollado por los laboratorios Thalmic™ (mostrado en la Figura 4.4) es un dispositivo en forma de brazalete que lee actividad eléctrica en los músculos usando sensores que ellos mismos desarrollan. Tiene la ventaja de no requerir el uso de gel conductivo ni afeitar el brazo. Este dispositivo es utilizado para la lectura de la actividad muscular del usuario para procesar los datos y controlar el prototipo desarrollado en este proyecto.

El brazalete contiene 8 sensores de señales EMG que representan el potencial eléctrico de los músculos como resultado de actividad muscular; es capaz de tener una frecuencia de muestreo de señales EMG de 200 Hz. Además, tiene nueve unidades de medición inercial axial (IMU) que contiene un giroscopio, un acelerómetro y un magnetómetro de 3 ejes cada uno.

El brazalete se comunica a través de la tecnología Bluetooth 4 (BLE), tiene un amplio soporte y sus herramientas SDK (Kit de Desarrollo de Software) la hacen muy accesible para el desarrollo de nuevos proyectos e investigaciones.

Actualmente sólo los sistemas operativos iOS y Windows pueden acceder a las señales EMG, sin embargo existen librerías de código abierto que pretenden hacer uso del brazalete mediante software desarrollado en lenguaje Pyhton, sin embargo tampoco se puede acceder a dichas señales en bruto sino a señales preprocesadas que facilitan su clasificación [30] [34].

Figura 4.4: Brazalete MYO™ [35].

4.4. Adquisición de datos del brazalete electromiográfico

El ordenador de placa reducida (SBC) de bajo costo Raspberry Pi®, es un ordenador de tamaño reducido desarrollado en Reino Unido por la fundación Raspberry Pi® (Universidad de Cambridge). El concepto es un ordenador que incluye unicamente los accesorios y/o periféricos que le permiten realizar su funcionamiento básico. Está formada por una placa que soporta varios componentes necesarios en un ordenador común y es capaz de comportarse como tal [36].

Una Raspberry Pi 3® es utilizada en el desarrollo del proyecto para lograr la obtención y procesamiento de datos del brazalete sin necesidad de utilizar una PC que restaría movilidad y portabilidad al prototipo.

Figura 4.5: Raspberry Pi 3® [37].

La Raspberry Pi 3® (mostrada en la Figura 4.5) es la tercera generación de placas, y contiene el siguiente hardware:

- ✓ Procesador Quad Core 1.2GHz Broadcom BCM2837 64bit.
- ✓ 1GB de Memoria RAM.
- ✓ Módulo inalámbrico LAN BCM43438.
- ✓ Módulo Bluetooth Low Energy (BLE).
- ✓ Conector GPIO de 40 pines.
- ✓ 4 puertos USB 2.
- ✓ Salida estéreo de 4 polos y un puerto de video compuesto.
- ✓ Puerto HDMI.
- ✓ Puerto Micro SD para carga de sistema operativo y unidad de almacenamiento.
- ✓ Puerto Micro USB para alimentación.

El conector GPIO cuenta con 40 pines de propósito general (donde algunos de ellos cumplen funciones de propósito específico), cuyo comportamiento (incluyendo si es un pin de entrada o salida) se puede controlar (programar) por el usuario en tiempo de ejecución [36] [37]. En la Figura 4.6 se muestra el diagrama de los puestos GPIO de la Raspberry Pi 3.

El sistema operativo oficial soportado por la Raspberry Pi 3 es Raspbian y está basado en el sistema operativo Debian. Por defecto, Python viene instalado en el sistema operativo de Raspbian, lo que lo hace factible para realizar una conexión con el brazalete electromiográfico mediante el script del proyecto myo-raw.

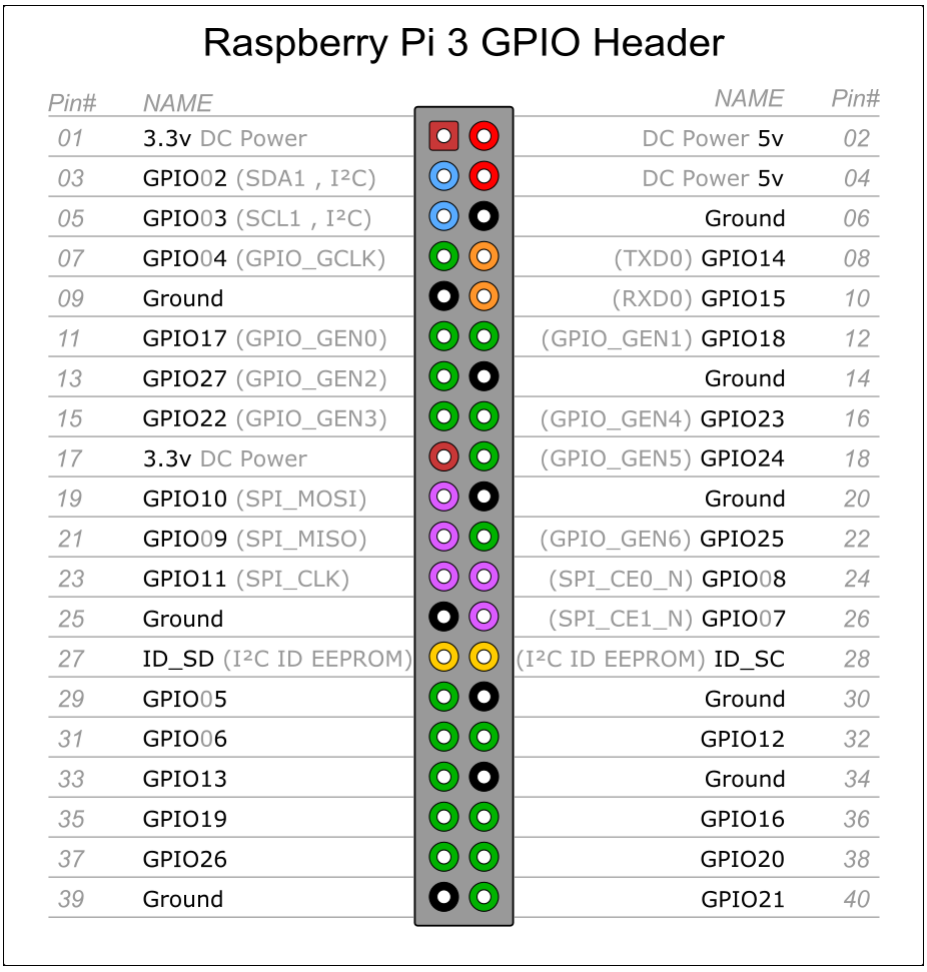


Figura 4.6: Diagrama de puertos GPIO de la Raspberry Pi 3.

El código permite la conexión, configuración y la obtención de datos de los sensores electromiográficos del brazalete mediante protocolos de comunicación Bluetooth 4.0. Para poder ejecutar dicho programa es necesario cumplir con ciertos requerimientos. Uno de ellos es que el brazalete haya sido conectado al menos una vez en una PC, además de que el sistema operativo en el cual se ejecutará el script deberá contar con:

- ✓ Una versión de Phyton 2.6 o mayor.
- ✓ El paquete pySerial instalado.
- ✓ El paquete enum34 instalado.
- ✓ El paquete pygame instalado.
- ✓ El paquete numpy instalado.

Para verificar que todo está instalado y se cumplen los requerimientos, se ejecutan los siguientes comandos en una terminal. El comando $python\ -V$ es utilizado para determinar la

versión de Python instalada en el sistema. En este caso se cuenta con la versión 2.7.13, como puede observarse en la Figura 4.7.

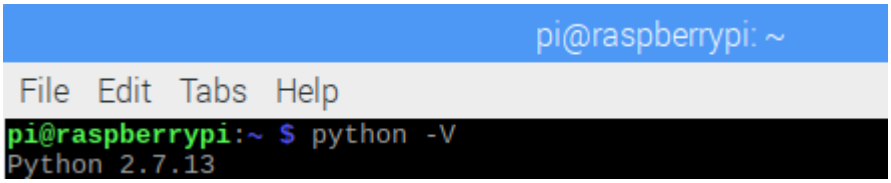


Figura 4.7: Versión de Python instalada por defecto en Raspbian.

El comando *pip list* es usado para mostrar una lista de los paquetes instalados en el sistema operativo que utiliza Python . En dicha lista deberán encontrarse todos los paquetes antes mencionados, como muestra la Figura 4.8.

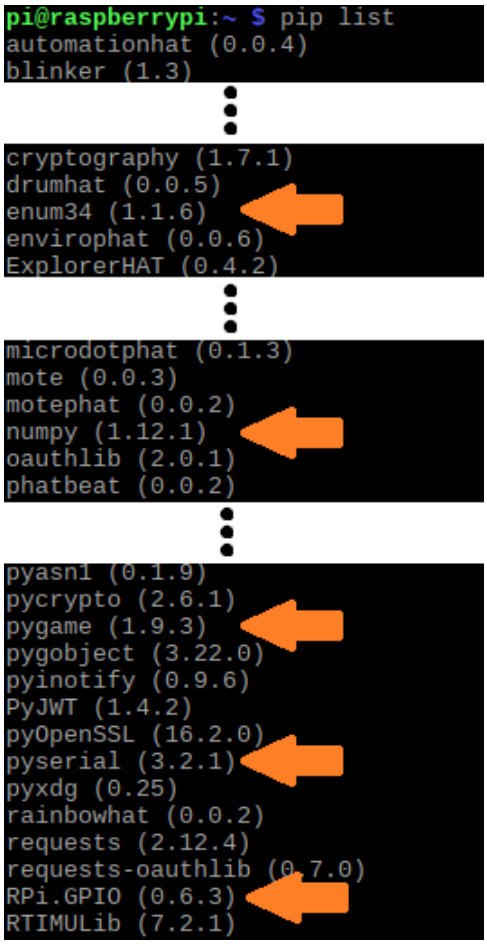


Figura 4.8: Paquetes requeridos instalados para el correcto funcionamiento del script.

Si alguno de los paquetes no está instalado se procede a instalarlo mediante el comando *sudo pip install* seguido del nombre del paquete a instalar (pySerial, enum34, pygame o numpy) según sea necesario.

Además, se debe verificar que el paquete RPi.GPIO esté instalado. Dicho paquete es necesario para controlar los pines GPIO a través del lenguaje Python y debería estar instalado por defecto.

Una vez verificados los requerimientos, se descargaron en la Raspberry Pi 3 los archivos *common.py* y *myo_raw.py* del proyecto desde la plataforma de GitHub [38]. Con el brazalete encendido y el adaptador Bluetooth conectado en los puertos USB de la Raspberry Pi 3 se ejecuta el script descargado para verificar su funcionamiento. Esto se hace mediante el comando *python myo_raw.py*. En la Figura 4.9 se observa el programa ejecutándose, el cual muestra una ventana con 8 gráficas representando los valores preprocesados que brinda cada sensor del brazalete.

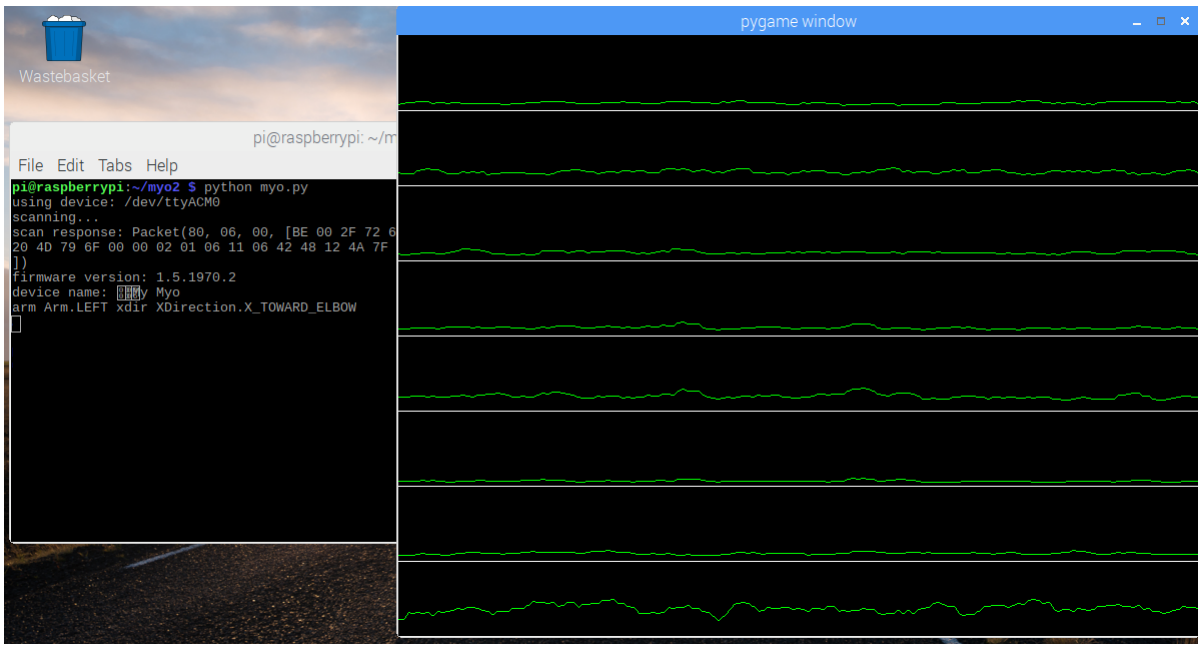


Figura 4.9: Script de Python ejecutándose mostrando gráficamente los valores obtenidos de cada sensor (gráficas de color verde).

Analizando el código fuente del archivo *myo_raw.py* se puede percibir que los datos preprocesados por el brazalete de los 8 sensores se guardan en la variable de tipo tupla llamada *emg*. Dichos valores son los que se emplearon para llevar a cabo su clasificación y de esta manera detectar la actividad muscular que controlará la mano robótica. Los valores contenidos en la variable *emg* son números enteros que van de 0 a 1023.

4.5. Adquisición de datos de los encoders

Para la lectura de los encoders y el cálculo de la posición de los dedos se utilizó un microcontrolador ATmega 328 (hoja de datos mostrada en el Apéndice B), el cual proporciona los pines de entrada/salida necesarios para el desarrollo del prototipo. En la Figura 4.10 se muestra el diagrama del microcontrolador.

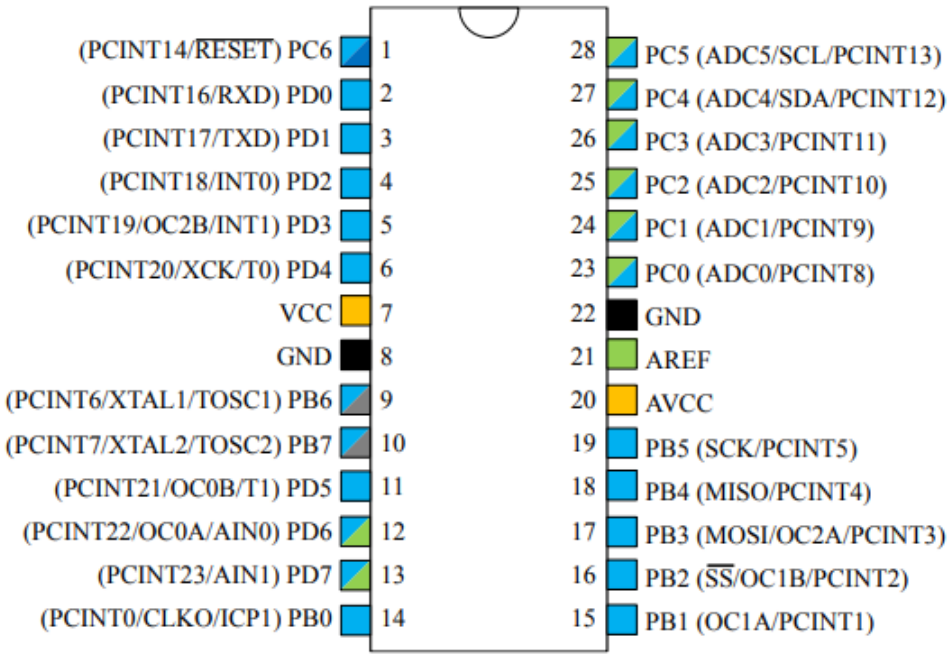


Figura 4.10: Diagrama del microcontrolador ATmega 328.

Este dispositivo es el encargado de contar los flancos obtenidos de cada uno de los encoders y determinar el número de grados que se ha movido el dedo. Con ayuda de la Tabla 4.1 se puede proponer una ecuación para convertir el número de flancos detectados en grados sexagesimales y determinar la posición de cada dedo. Para la apertura y cierre de los dedos se plantea:

$$G = \frac{n_f(360)}{(150)(4{,}5)(6)} \tag{4.5.1}$$

y para la rotación del dedo pulgar:

$$G = \frac{n_f(360)}{(150)(14)(6)} \tag{4.5.2}$$

Donde G es la posición del dedo en grados y n_f es el número de flancos generados por el encoder. Hay que resaltar que el número de flancos se deben contar desde una posición inicial y dependiendo del sentido de giro se suman o se restan. Así, si el micromotor gira en sentido positivo generando 100 flancos y enseguida gira en sentido opuesto generando 30 flancos, el número de flancos a utilizar en la ecuación será de 70. Debido a que al utilizar sólo un canal de salida del encoder no se puede saber el sentido de giro del micromotor, éste será definido por el pin de control del controlador.

4.6. Sistema electrónico

El sistema electrónico se muestra en la Figura 4.11. Puede observarse que elementos como los encoders y controladores de los motores son conectados al microcontrolador. La Raspberry Pi 3 también está conectada a él a través de sus pines GPIO, al mismo tiempo que se transmite información con el brazalete por medio del adaptador Bluetooth 4.0.

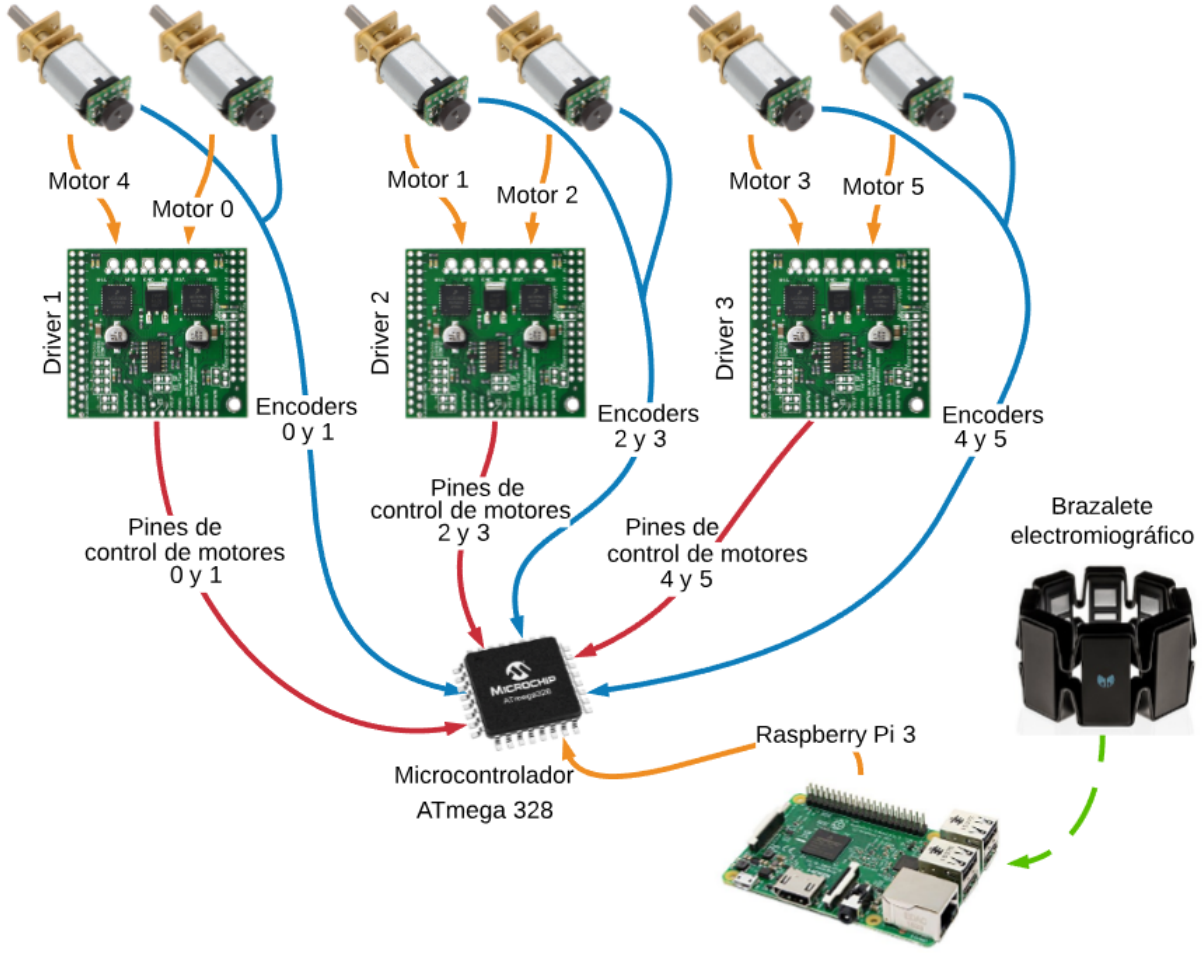


Figura 4.11: Sistema electrónico utilizado para el control de la mano robótica.

El voltaje utilizado para la alimentación del microcontrolador, drivers y encoders es de 3.3V regulados mediante un LM7833 (hoja de datos mostrada en el Apéndice C), obtenidos de una fuente de alimentación de 5V con la cual se alimentan los motores (Etapa de potencia de los controladores). Los 5V son proporcionados por una fuente de corriente directa, pudiendo ser sustituida por una batería u otra fuente que proporcione el voltaje y la corriente necesaria para funcionar (de 5V a 18V a 1.5A).

En la Tabla 4.2 se muestran y describen las conexiones existentes entre los diferentes elementos del sistema y el microcontrolador.

Conexiones del microcontrolador			
No. Pin	Pin	Elemento	Descripción
2	PD0	GPIO02	Comunicación con la Raspberry Pi 3
3	PD1	GPIO03	Comunicación con la Raspberry Pi 3
4	PD2	GPIO04	Comunicación con la Raspberry Pi 3
5	PD3/ OC2B	M2DIR (Driver 3)	Pin de control de velocidad (PWM) para el motor 5
6	PD4	GPIO17	Comunicación con la Raspberry Pi 3
7	VCC	+5V	Alimentación de 5 volts
8	GND	GND	Tierra
9	PB6	OUTA (Encoder 2)	Tren de pulsos
10	PB7	OUTA (Encoder 3)	Tren de pulsos
11	PD5/ OC0B	M1DIR (Driver 2)	Pin de control de velocidad (PWM) para el motor 1
12	PD6/ OC0A	M2DIR (Driver 1)	Pin de control de velocidad (PWM) para el motor 0
13	PD7	OUTA (Encoder 5)	Tren de pulsos
14	PB0	OUTA (Encoder 4)	Tren de pulsos
15	PB1/ OC1A	M2DIR (Driver 2)	Pin de control de velocidad (PWM) para el motor 2
16	PB2/ OC1B	M1DIR (Driver 3)	Pin de control de velocidad (PWM) (PWM) para el motor 3
17	PB3/ OC2A	M1DIR (Driver 1)	Pin de control de velocidad (PWM) (PWM) para el motor 4
18	PB4	OUTA (Encoder 0)	Tren de pulsos
19	PB5	OUTA (Encoder 1)	Tren de pulsos
23	PC0	M2DIR (Driver 1)	Pin de control de dirección para el motor 0
24	PC1	M1DIR (Driver 2)	Pin de control de dirección para el motor 1
25	PC2	M2DIR (Driver 2)	Pin de control de dirección para el motor 2
26	PC3	M1DIR (Driver 3)	Pin de control de dirección para el motor 3
27	PC4	M1DIR (Driver 1)	Pin de control de dirección para el motor 4
28	PC5	M2DIR (Driver 3)	Pin de control de dirección para el motor 5

Tabla 4.2: *Elementos conectados al microcontrolador Atmega 328*

Capítulo 5

Control de la mano robótica

El sistema de control protésico se dividió en dos subsistemas, uno encargado de la detección y clasificación de la actividad muscular con los datos obtenidos del brazalete y el otro encargado de la ejecución de las estrategias de agarre en la mano robótica. Los subsistemas están controlados por la Raspberry Pi y el microcontrolador ATmega 328 respectivamente, interconectados por 4 pines con los cuales la Raspberry Pi envía al microcontrolador la información necesaria para determinar el movimiento a realizar por la mano robótica.

A continuación se expone el desarrollo del código implementado en la Raspberry Pi 3 para la clasificación y detección de gestos a través del brazalete, además de la implementación del código en el microcontrolador, encargado de la activación de los motores para realizar las estrategias de agarre en la mano robótica.

5.1. Subsistema brazalete electromiográfico - Raspberry Pi 3

La lógica que rige a este subsistema se puede observar en el diagrama de bloques de la Figura 5.1. En general, su comportamiento se describe a continuación. Como paso inicial se realiza la adquisición de los datos recolectados por el brazalete (comentada en la Sección 4.4). Los datos son capturados en la variable *emg*, la cual se exporta a un módulo de extensión en C para llevar a cabo la clasificación. La implementación del módulo de extensión se realizó con el fin de aplicar la clasificación de patrones de valor real basada en MAE en lenguaje C para facilitar su traslado a otros sistemas (como un microcontrolador), además de ejecutarse a mayor velocidad que en lenguaje Python.

Una vez realizada la clasificación y obtenido el índice de la clase, se retorna dicho valor al *script* principal, para posteriormente transmitir el dato al microcontrolador. En el Apéndice D se anexa el código modificado del script en Python encargado de realizar la comunicación y configuración con el brazalete mioeléctrico, así como la exportación de datos de los sensores a la extensión en C y la comunicación con el microcontrolador. En el Apéndice E se muestra el código en C de la extensión en la cual se ejecuta la etapa de clasificación y se retorna el valor de la clase definida al script de Python.

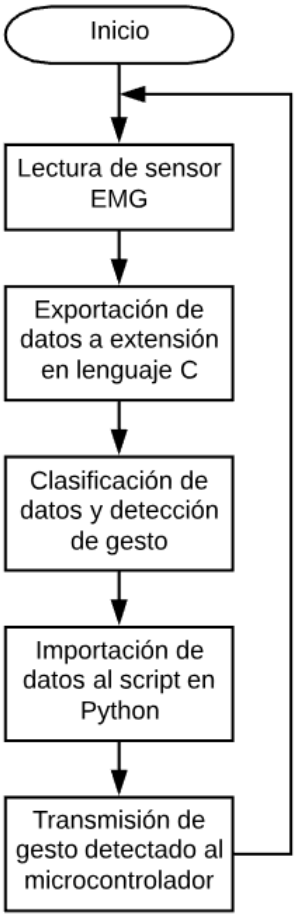


Figura 5.1: Comportamiento del sistema Brazalete-Raspberry Pi.

5.1.1. Fase de entrenamiento y clasificación

Para poder llevar a cabo la fase de entrenamiento y obtener la matriz $\mathbf{M}$ de la ecuación (2.4.1) se deben considerar tanto los gestos a realizar con la mano robótica como los gestos a identificar con el brazalete. Los gestos a implementar en la mano robótica son mostrados en la Figura 5.2. En dicha figura se presentan las 6 estrategias de agarre más comunes en la mano humana [23] y dos gestos añadidos para facilitar el agarre de objetos y brindar una posición inicial a los dedos (gesto de palma abierta y gesto de puño, respectivamente). Tomando en cuenta las configuraciones de agarre a ejecutar con la mano robótica, se necesitan 8 clases de patrones identificables por el sistema brazalete-Raspberry Pi. Dichos patrones se muestran en la Figura 5.3.

Una vez considerado el número de clases a usar, se procedió a evaluar la función ϕ_i utilizando el operador **prom**, ecuación (2.4.2). Debido a que el brazalete cuenta con 8 sensores, el índice j de dicha función es $j = 1, 2, 3, 4, 5, 6, 7, 8$. Para obtener los coeficientes de la matriz $\mathbf{M}$ se evaluó la función *phi* con 100 datos por sensor para cada uno de los patrones a identificar.

Figura 5.2: Gestos a realizar por la mano robótica.

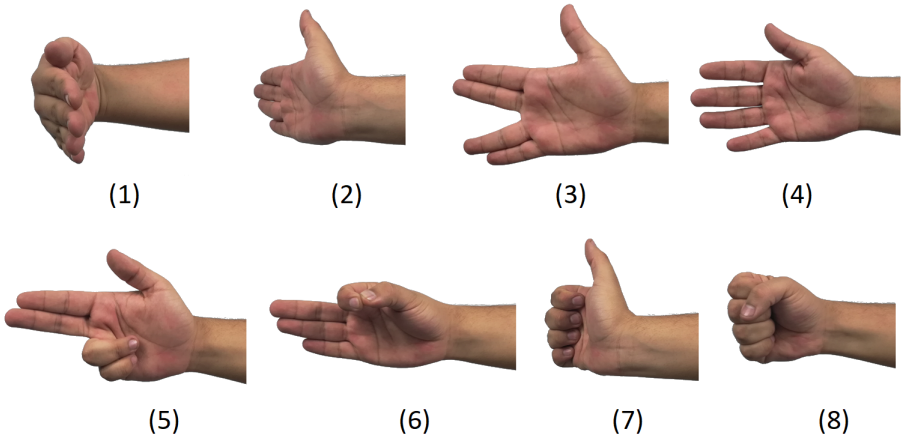


Figura 5.3: Patrones a identificar por medio del brazalete.

De esta forma se obtiene la matriz de 8×8:

$$\mathbf{M} = \begin{bmatrix} 20 & 22 & 25 & 74 & 35 & 38 & 26 & 23 \\ 189 & 208 & 110 & 185 & 441 & 196 & 189 & 525 \\ 621 & 287 & 96 & 130 & 123 & 162 & 498 & 271 \\ 88 & 146 & 453 & 903 & 511 & 113 & 73 & 75 \\ 259 & 127 & 246 & 609 & 236 & 78 & 211 & 147 \\ 182 & 117 & 133 & 273 & 157 & 75 & 208 & 603 \\ 35 & 56 & 95 & 289 & 164 & 64 & 52 & 51 \\ 74 & 209 & 585 & 471 & 364 & 174 & 72 & 75 \end{bmatrix} \tag{5.1.1}$$

en la cual se muestran por filas, los gestos correspondientes en el orden indicado con la numeración de la Figura 5.3.

Teniendo la matriz de memoria asociativa ($\mathbf{M}$) se procede con la fase de clasificación. En esta fase se evalúan los datos leídos actualmente por el brazalete en la ecuación (2.4.5) y

se determina el índice de la clase a la que pertenece. Es entonces que la fase de transmisión comienza.

5.1.2. Transmisión de datos al microcontrolador

La transmisión de datos se realizó utilizando un sistema multiplexor usando los pines GPIO02, GPIO03, GPIO04 y GPIO17 de la Raspberry Pi 3 y los pines PD0, PD1, PD2, PD4 del microcontrolador de acuerdo a los valores asignados en la Tabla 5.1. De esta forma, al colocar un estado lógico específico en cada pin de la Raspberry, el microcontrolador identificará el índice detectado por la SBC y se ejecutará el gesto correspondiente en la mano robótica.

Pin del SBC	Pin del ATmega	Estado lógico							
GPIO02	PD00	0	1	0	1	0	1	0	1
GPIO03	PD01	0	0	1	1	0	0	1	1
GPIO04	PD02	0	0	0	0	1	1	1	1
GPIO17	PD03	1	1	1	1	1	1	1	1
Índice detectado		1	2	3	4	5	6	7	8

Tabla 5.1: *Relación entre estados lógicos e índice de clasificación detectado*

Como puede observarse, el estado del pin GPIO017 siempre debe estar en alto para identificar un índice, por lo que funciona como un habilitador que permite indicar al microcontrolador que la clasificación se está llevando a cabo, y así evitar que se detecte un gesto por parte del microcontrolador mientras la Raspberry Pi 3 se está inicializando.

5.2. Subsistema microcontrolador-mano robótica

Una vez identificado el patrón, el microcontrolador ATmega328 es el encargado de verificar que la posición de cada uno de los dedos se encuentre en el ángulo que corresponda para realizar el gesto deseado, y activar los micromotores según sea necesario. Dichos ángulos variarán de acuerdo al objeto a sujetar. En la Figura 5.4 se muestra el diagrama de flujo que representa el comportamiento del microcontrolador, mientras que en el Apéndice F se muestra el código fuente que ejecuta.

Para realizar cualquier gesto, se va evaluando dedo por dedo y se mueve a la posición adecuada dependiendo del gesto requerido. Esto se realiza un motor a la vez, uno tras otro para utilizar únicamente una fuente de alimentación para todos los movimientos, impidiendo que se activen dos motores a la vez por las limitaciones de corriente que la fuente pueda tener (tomando en cuenta la máxima corriente requerida por cada micromotor con carga máxima es de 3 A).

En un principio, la mano robótica siempre se iniciará con el gesto de puño realizado (la posición de cada dedo será conocida) y posteriormente se comparará dicho gesto con el que la Raspberry Pi 3 le indique realizar. La variable *motor* indicará el número de motor a controlar

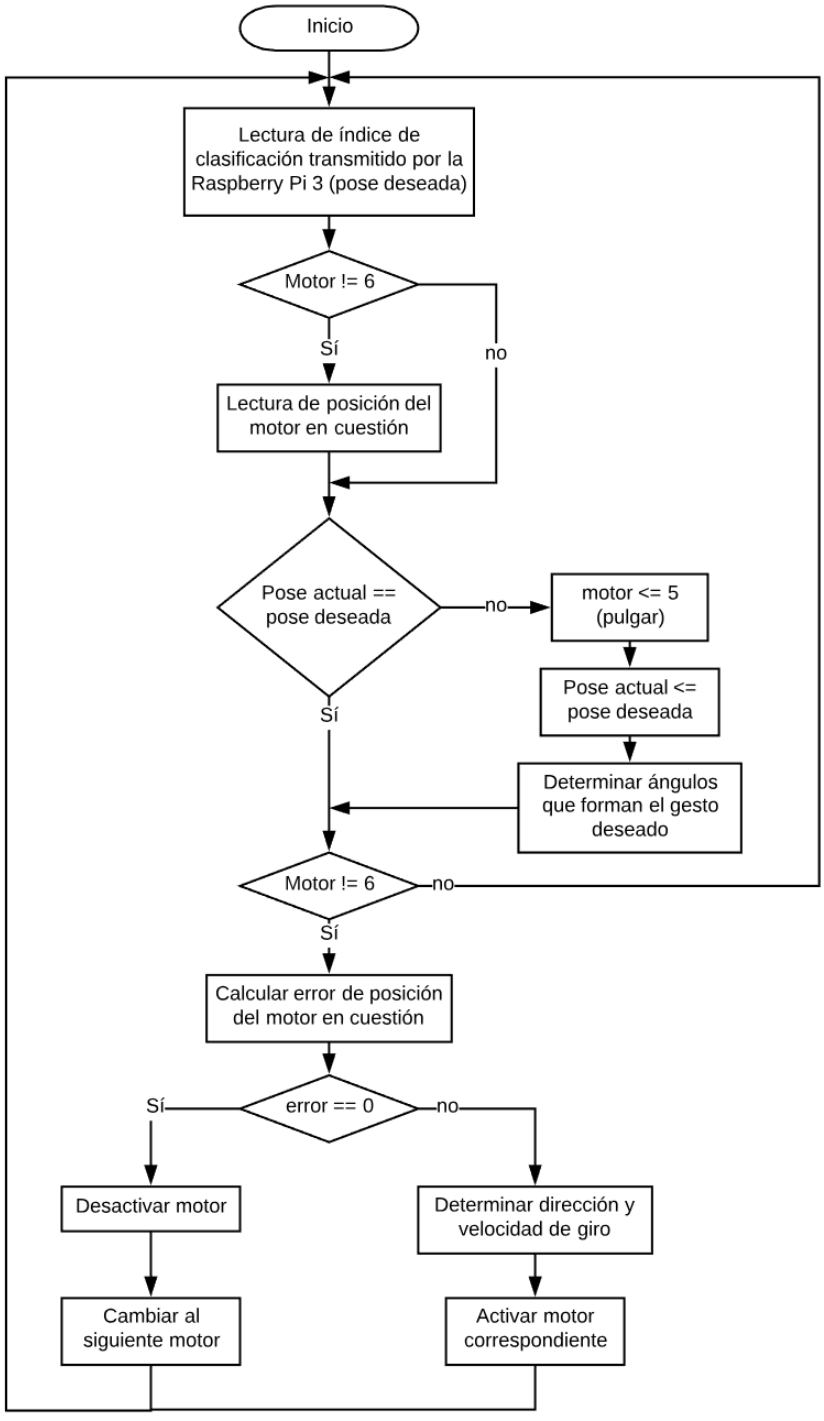


Figura 5.4: Diagrama de flujo del comportamiento del microcontrolador.

y será de ayuda para determinar el orden de movimiento de los dedos, con el fin de que no existan interferencias entre sí o con el objeto al momento de cambiar de estrategia de agarre. Inicialmente el valor de esta variable es 6 e indica que no se trabajará con ningún motor.

Los valores de esta variable van del 0 al 6, correspondiendo el número 0 al motor del dedo meñique, el 1 al dedo anular, el 2 al dedo medio, el 3 al dedo índice, el 4 al motor de rotación del pulgar y el 5 al pulgar.

Como primer paso, se obtiene el índice de clasificación que determina el gesto a realizar (mediante la lectura de los pines PD0, PD1, PD2 y PD4 y basándose en la Tabla 5.1). Cada índice de clasificación corresponderá a una estrategia de agarre definida previamente con los ángulos en los que se deben posicionar los dedos para formar el gesto. Una vez determinadas las posiciones a las cuales se deben colocar los dedos, se verifica el motor con el que se trabajará y en caso de que su valor sea distinto a 6, se leerá la posición actual del motor indicado. En el caso $motor == 6$ simplemente se omitirá la lectura de posición.

Una vez realizado esto, se verifica que el gesto realizado sea el mismo que el que se desea realizar (indicado por el índice de clasificación). En caso de ser diferente, indica que un cambio de gesto se ha realizado y se modifica el valor de la variable *motor* para comenzar el movimiento de los dedos. También se actualiza el valor de la pose actual por el del nuevo gesto y con ayuda de una base de datos con las posiciones que conforman cada gesto, se obtienen los ángulos en los cuales se deberán posicionar los dedos. Si la pose deseada concuerda con la pose actual se omiten los pasos anteriormente descritos.

Enseguida se verifica nuevamente el motor con el que se trabajará y en caso de ser $motor == 6$ se vuelve al inicio del programa. Caso contrario, se procede a calcular el error de posición del motor indicado mediante una resta:

$$error = posición\ actual - posición\ deseada$$

De esta manera se obtiene el valor en grados que deberá moverse el micromotor. Si el valor del error es distinto a 0, se determina la velocidad del micromotor en base a su magnitud, y el sentido de giro dependiendo de su signo (el signo negativo indicará la necesidad de abrir el dedo).

La velocidad, controlada por las señales PWM del microcontrolador, variará como se muestra en la Tabla 5.2. Estos cambios de velocidad se implementaron para reducir la velocidad del movimiento del dedo cuando el error es menor y de esta manera facilitar el frenado del micromotor, ya que al tratar de frenarlo desde una velocidad mayor, el momento de inercia generado provoca que el micromotor gire más allá de la posición indicada, causando que el gesto se realice con menor precisión. Cuando el valor de error sea $error == 0$ indicará que el

Valor del error	Porcentaje de velocidad (%)	Valor de PWM (0-255)
error >= 15°	100 %	255
15° >error >= 5°	58.8 %	150
error <5°	29.4 %	75

Tabla 5.2: *Relación entre el error de posición y la velocidad del micromotor*

dedo ha llegado a su posición deseada y se deberá desactivar el movimiento del micromotor en cuestión y continuar con el siguiente motor correspondiente.

En la Figura 5.5 se muestra el orden en el que la variable *motor* se modifica. Inicialmente el valor está predefinido en 6. Cuando se detecta un cambio de gesto en el brazalete, la

variable *motor* toma el valor de 5. A causa de ser el primer movimiento para cambiar de gesto, se programa al sistema para que realice un movimiento de apertura del dedo, lo que permitirá evitar que sus movimientos interfieran entre sí y soltar un objeto previamente sujeto. Al abrirse el dedo pulgar, el valor de la variable cambia a 4 con lo cual se rota el pulgar a la posición deseada. Enseguida $motor = 0$ y el programa moverá el dedo meñique a su posición, seguido de los motores 1, 2 y 3 en ese orden. Después de eso, el valor cambia a 5 para realizar el último movimiento que sujetará al objeto: cerrar el dedo pulgar a la posición deseada. Al ser el último movimiento para formar la configuración de agarre, el valor vuelve a ser 6, por lo que no volverá a modificarse hasta ser detectada una nueva pose.

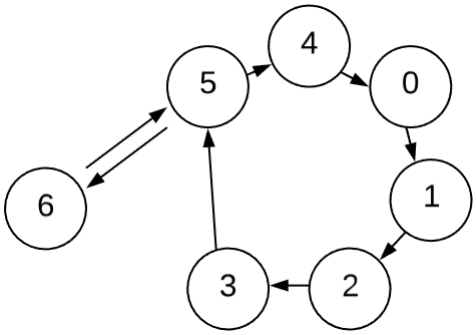


Figura 5.5: Sucesión de valores para la variable *motor*.

5.3. Estimación de posiciones para realizar las estrategias de agarre

Para ejecutar con la mano robótica las estrategias de agarre propuestas, es necesario conocer el ángulo de cada dedo en cada una de las estrategias a implementar. Para ello se realizaron dibujos de algunos objetos con diferentes formas que permitieran a la mano hacer uso de las distintas estrategias de agarre para sujetarlos y se simularon las posturas de la mano sujetando dichos objetos y de esta forma estimar las posiciones de cada dedo para cada agarre. En la Figura 5.6 se puede observar la representación del agarre cilíndrico tomando uno de los objetos simulados (un vaso) y en la Figura 5.7 se exponen los ángulos estimados para el dedo índice para el mismo agarre. Como puede observarse, los ángulos recorridos por ambas falanges del dedo (proximal y distal) es el mismo.

Los ángulos estimados para cada postura se registraron en la Tabla 5.3. En ella se muestran las posiciones de las falanges proximales de cada dedo (dedos opuestos al pulgar). La primera columna de ángulos pertenecientes a la sección del dedo pulgar corresponden al ángulo de la falange proximal (movimiento independiente) y la segunda columna corresponde al ángulo de la falange distal (movimiento dependiente). Éste dato ha sido omitido en la tabla para los demás dedos debido a que el ángulo es el mismo en ambas falanges.

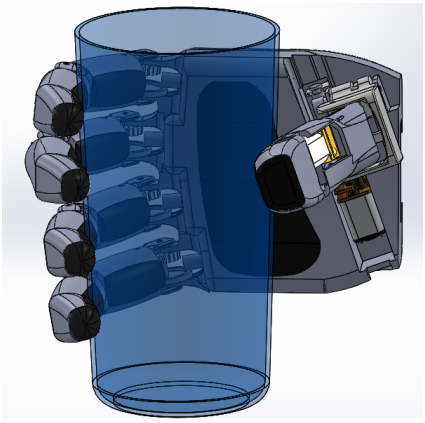


Figura 5.6: Simulación del agarre cilíndrico sujetando un vaso.

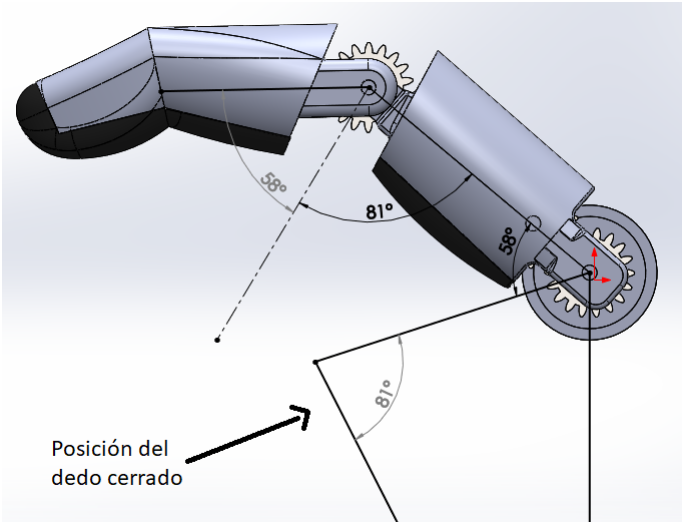


Figura 5.7: Estimación del ángulo del dedo índice para realizar el agarre cilímdrico.

Gesto	Meñique	Anular	Medio	Índice	Palma	Pulgar	
Cilindrico	61°	56°	53°	58°	15°	64°	32°
Punta	0°	0°	0°	50°	50°	26°	13°
Gancho	32°	28°	22°	30°	110°	90°	45°
Abierta	106°	10°	106°	106°	110°	90°	45°
Palmar	0°	0°	38°	47°	40°	32°	26°
Esférico	34°	46°	42°	33°	15°	54°	27°
Lateral	0°	11°	21°	32°	100°	18°	9°
Puño	0°	0°	0°	0°	0°	15°	7,5°

Tabla 5.3: *Ángulos de la mano al realizar las distintas estrategias de agarre*

5.4. Funcionamiento del sistema integrado

En la Figura 5.8 se muestra el diagrama a bloques que representa al sistema completo. En general, su funcionamiento se puede describir de acuerdo a las funciones que cumple cada uno de los elementos que lo conforman: El brazalete electromiográfico, como su nombre lo indica, es el encargado de obtener las señales electromiográficas de los músculos del brazo del usuario y realizar un procesamiento a los datos, previo a su transmisión por medio de tecnología Bluetooth.

La Raspberry Pi 3 tiene la función de comunicarse con el brazalete y accesar a la información transmitida. De esta manera se obtienen las señales de cada uno de los 8 sensores EMG y se realiza la clasificación. El gesto, una vez identificado, es transmitido al microcontrolador por medio de los puertos GPIO de la Raspberry Pi 3.

El microcontrolador recibe la información del gesto detectado y mediante una base de datos obtiene la posición en la que debe estar cada dedo de la mano robótica (posición deseada) para ejecutar la estrategia de agarre correspondiente. A través de un pin de salida y una señal PWM, el microcontrolador le indica a cada uno de los controladores de los motores la velocidad y el sentido de giro con el cual deberán moverse.

El motor genera el movimiento y lo transmite a los mecanismos de la mano, produciendo el movimiento de los dedos.

El encoder, al estar acoplado al eje del micromotor comienza a girar y genera una señal de tren de pulsos. Esta señal se analiza en el microcontrolador y se calcula el movimiento generado por el dedo. De esta forma el sistema se retroalimenta con la posición actual de cada motor y la compara con la posición deseada, adaptando la velocidad y dirección de los actuadores según sea requerido.

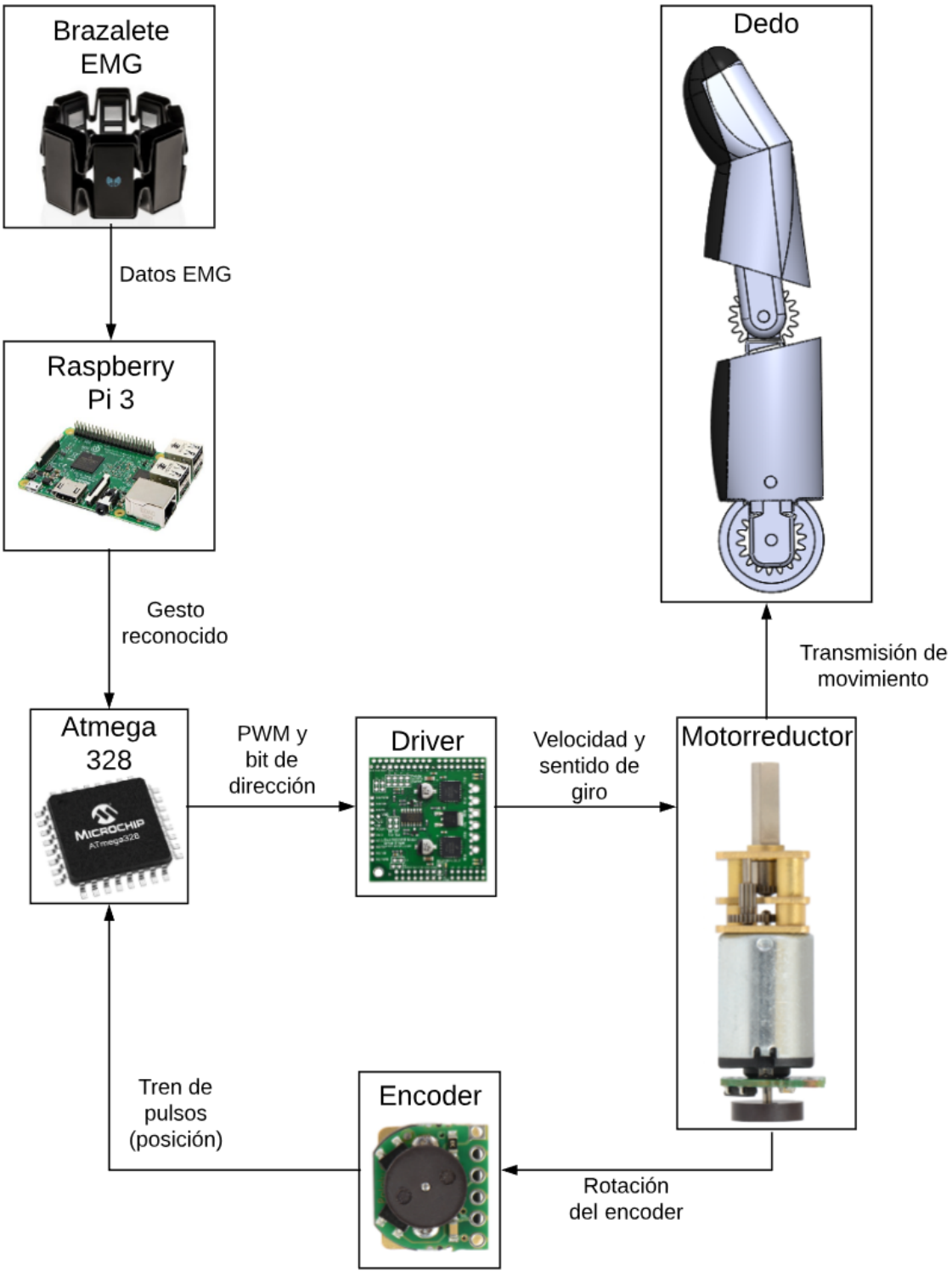


Figura 5.8: Diagrama de funcionamiento del sistema completo.

Capítulo 6

Pruebas y resultados

En este capítulo se muestran las pruebas realizadas para evaluar el funcionamiento del prototipo y los resultados obtenidos.

Las pruebas realizadas son las siguientes:

- ✓ Prueba de desempeño de la clasificación
- ✓ Prueba de posicionamiento de los dedos al realizar agarres

6.1. Prueba de desempeño de la clasificación de datos

La primer prueba consistió en verificar y calcular con porcentajes, el desempeño de la clasificación realizada, ejecutando todos los gestos diez veces y cuantificando las ocasiones en las que la clasificación se llevó a cabo satisfactoriamente, es decir, que la estrategia de agarre realizada por la mano robótica corresponda al gesto realizado. Mediante la realización de ésta prueba se verifica el funcionamiento del brazalete, su comunicación con la Raspberry Pi 3 y los procesos de clasificación realizados.

Se realizaron dos pruebas de éste tipo, con la diferencia de que la primera se llevó a cabo tras realizar la fase de entrenamiento del sistema clasificador sin mover la posición y ubicación del brazalete, y la segunda prueba se realizó en un momento distinto tras retirar el brazalete del brazo y luego colocarlo en el mismo sitio en el que se realizó el entrenamiento.

En la Tabla 6.1 se pueden observar los resultados obtenidos de cada una de las 10 repeticiones (columna "Número de ejecución") de la primera prueba realizada, indicando en cada una de las casillas el gesto realizado por la mano robítica.

La columna "Desempeño" muestra en porcentaje el número de veces que se realizaron las estrategias de agarre correctamente en el momento requerido.

Los resultados mostrados en la Tabla 6.2 son los correspondientes a la segunda repetición de la prueba.

Gesto realizado	Estrategia de agarre	Número de ejecución										Desempeño (%)
		1	2	3	4	5	6	7	8	9	10	
(1)	Cilíndrico (C)	C	C	C	C	C	C	C	Pr	C	C	90 %
(2)	Punta (P)	P	G	P	G	P	P	P	P	G	G	60 %
(3)	Gancho (G)	G	G	G	G	G	E	G	E	E	G	70 %
(4)	Abierta (A)	A	A	A	A	A	A	A	A	A	A	100 %
(5)	Palmar (Pr)	Pr	Pr	Pr	Pr	Pr	Pr	Pr	Pr	Pr	Pr	100 %
(6)	Esférico (E)	E	E	E	E	E	E	E	E	E	E	100 %
(7)	Lateral (L)	L	L	L	L	L	L	L	L	L	E	90 %
(8)	Puño (Po)	Po	Po	Po	Po	Po	Po	Po	Po	Pr	Pr	80 %

Tabla 6.1: *Desempeño del modelo **MAE** para la clasificación de datos*

Gesto realizado	Estrategia de agarre	Número de ejecución										Desempeño (%)
		1	2	3	4	5	6	7	8	9	10	
(1)	Cilíndrico (C)	C	C	C	C	C	C	C	C	C	C	100 %
(2)	Punta (P)	P	P	G	G	G	P	G	P	G	G	40 %
(3)	Gancho(G)	E	E	E	E	E	E	E	G	E	E	10 %
(4)	Abierta (A)	A	A	A	A	A	A	A	A	A	A	100 %
(5)	Palmar (Pr)	Pr	Pr	Pr	Pr	Pr	Pr	Pr	Pr	Pr	Pr	100 %
(6)	Esférico (E)	E	E	E	E	E	E	E	E	E	E	100 %
(7)	Lateral (L)	-	G	G	G	-	G	-	-	G	L	10 %
(8)	Puño (Po)	Po	Po	Po	Po	Po	Pr	Po	Po	Po	Pr	80 %

Tabla 6.2: *Desempeño del modelo **MAE** para la clasificación de datos de la segunda prueba*

6.1.1. Análisis de resultados

Las pruebas realizadas en cada ejecución mostraron que al realizar el gesto (2) para la estrategia de agarre tipo punta (P), se ejecutaba por momentos el gesto adecuado intercambiando momentáneamente con el gesto de agarre de tipo gancho (G). De la misma manera ocurrieron clasificaciones erróneas con la evaluación de los gestos de agarre tipo gancho realizando una estrategia de agarre esférico (E). Al evaluar el gesto de puño (Po) se ejecutó equivocadamente en dos ocasiones el agarre palmar (Pr). Al realizar el gesto correspondiente al agarre lateral (L), la mano robótica se colocó en posición para realizar el agarre esférico (E) y al realizar el gesto (1) se ejecutó el agarre palmar (Pr), fallando ambas pruebas en una sola ejecución. El resultado de todas las ejecuciones se puede observar en la Tabla 6.1.

Con la ejecución de ésta prueba se calculó el procentaje del desempeño general del prototipo, correspondiéndole un valor promedio del 86,25 %.

La segunda repetición de la prueba, tras remover y volver a colocar el brazalete, resultó con un porcentaje de desempeño de 67,5 % calculado de la misma manera que en la primera prueba.

6.2. Prueba de posicionamiento de los dedos

Este ejercicio consistió en verificar que todos los dedos se posicionen en el lugar apropiado para ejecutar el gesto de agarre correspondiente. Para ello fue necesario capturar la señal proporcionada por los encoders al realizar cada gesto y cuantificar el número de cambios de estado presentes en las señales para conocer la posición de los motores (y por consiguiente de los dedos). La captura de datos fue realizada utilizando una tarjeta de adquisición de datos e importando la información al software MatLab. Se realizó el conteo de flancos en cada una de las señales capturadas y dichos números se evaluaron (para cada instante del tiempo de captura) en las ecuaciones (4.5) y (4.5.2) según correspondiera la señal. De ésta manera se obtuvieron las posiciones de cada uno de los dedos y se realizaron las gráficas de posición angular vs tiempo correspondientes.

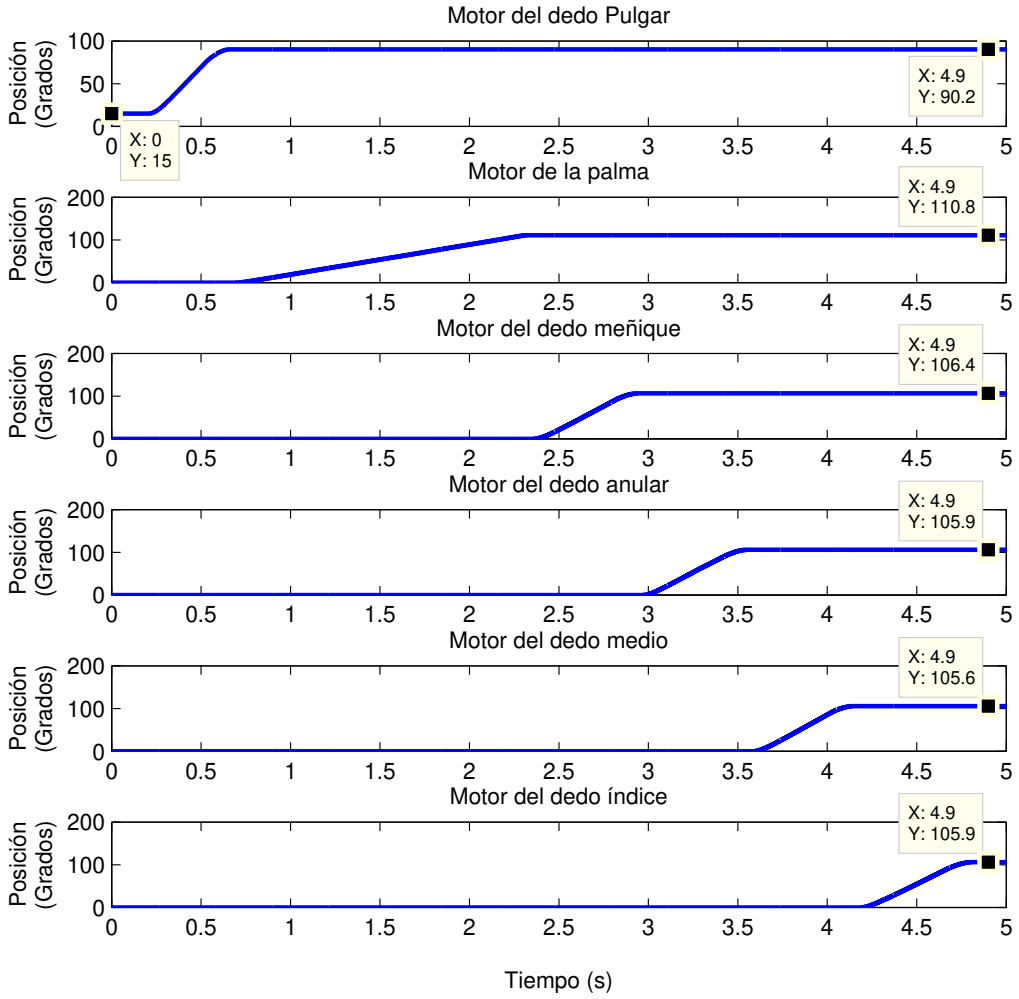


Figura 6.1: Posición de los dedos al abrirse la mano robótica partiendo del puño.

El primer gesto realizado fue la mano abierta, partiendo desde la posición inicial (gesto

de puño). Todos los gestos realizados posteriormente partieron de la posición de mano abierta, ya que es el gesto más adecuado para ejecutar estrategias de agarre sin interferir con el objeto a sujetar mientras se producen los movimientos. En las Figuras 6.1, 6.2 y 6.3 se muestran las gráficas de las posiciones de cada uno de los dedos durante la realización de diferentes gestos.

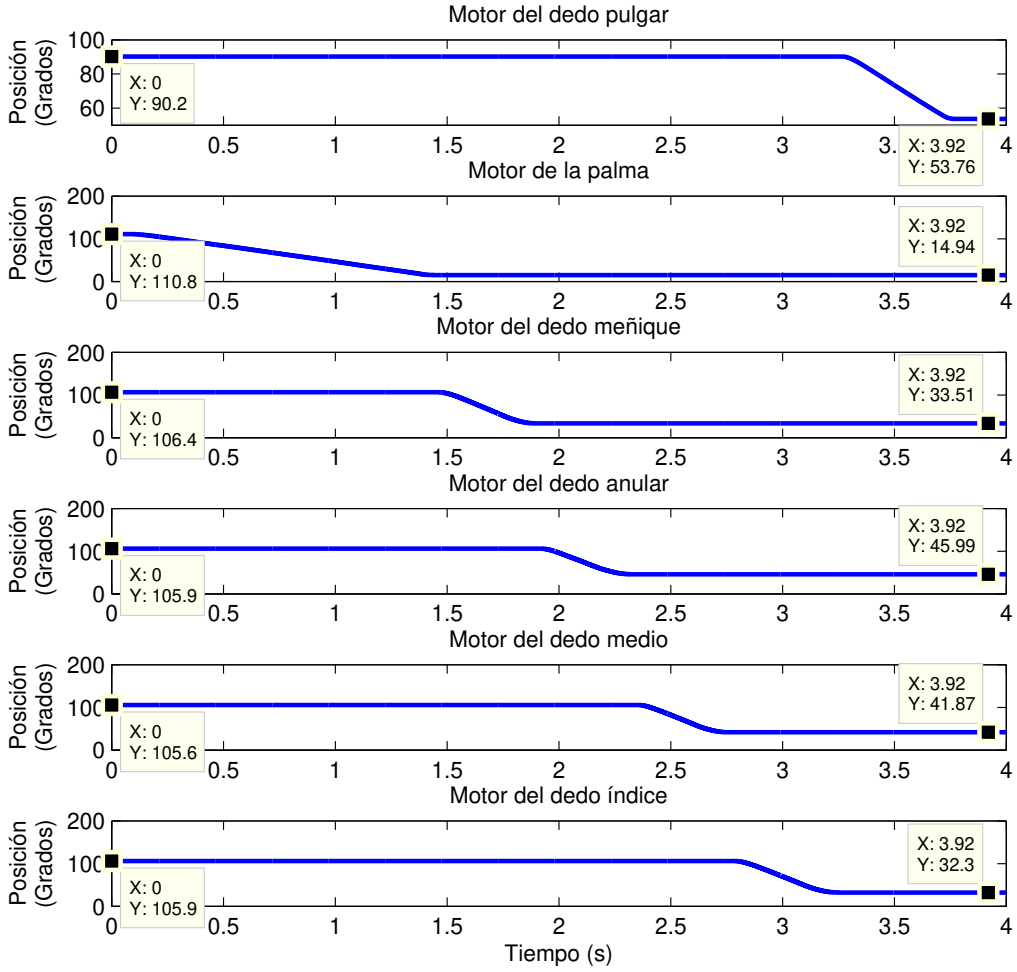


Figura 6.2: Posición de los dedos al ejecutar el agarre esférico partiendo de la mano abierta.

La Figura 6.1 muestra el movimiento de la mano robótica al abrirse, partiendo del gesto puño. Comenzando de ésta posición (mano abierta) se realiza la estrategia de agarre esférica. Las posiciones registradas para éste gesto se pueden observar en la Figura 6.2. Para realizar un nuevo gesto, se regresa a la posición abierta de la mano (Figura 6.3) y se procede con otro gesto. De esta manera se lleva a cabo la recopilación de información para todos los gestos.

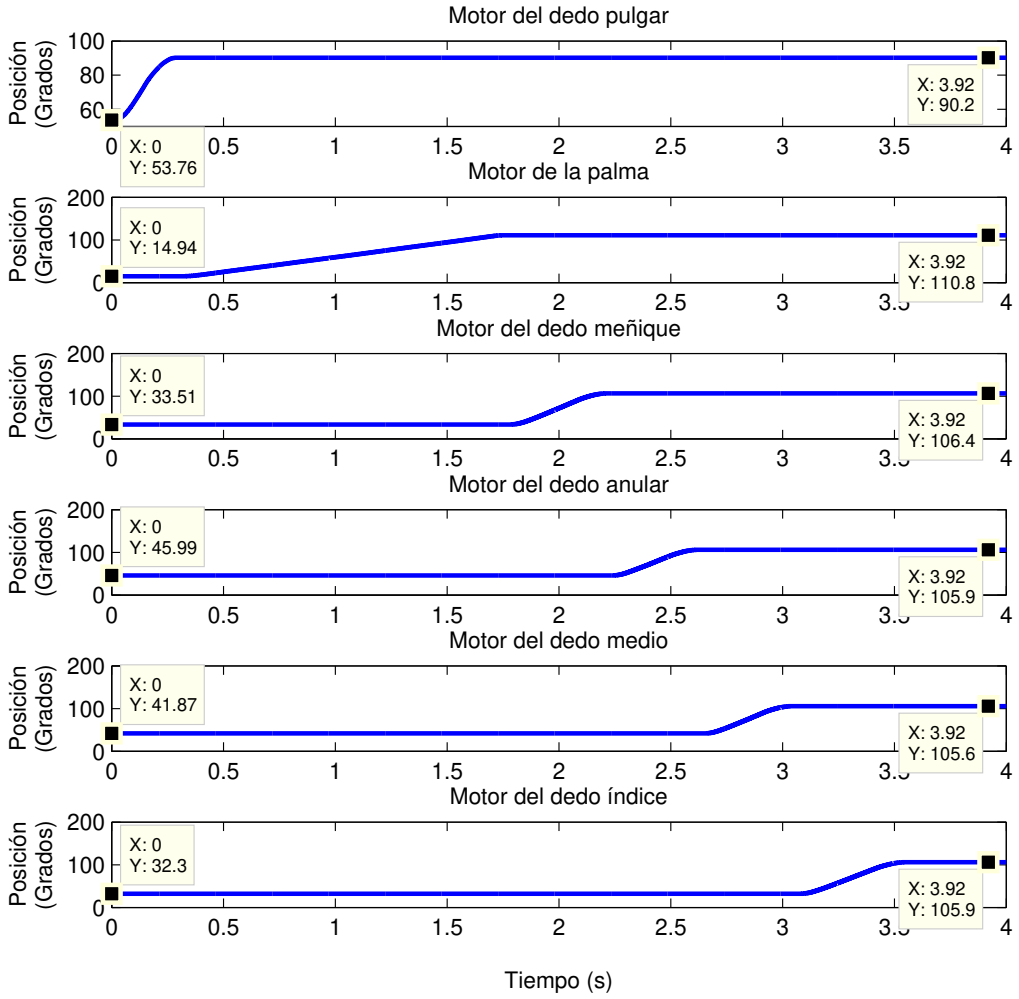


Figura 6.3: Posición de los dedos al regresar a la mano abierta desde el agarre cilíndrico.

Una vez conocida la posición real de cada dedo (en cada estrategia de sujeción), la información se compara con los ángulos deseados para determinar si el agarre se está ejecutando correctamente. La comparación de las posiciones de los dedos meñique, anular y medio se muestran en la Tabla 6.3. La comparación de ángulos de los dedos indice, palma, pulgar se exponen en la Tabla 6.4.

La primer columna de ángulos de cada dedo muestra las posiciones estimadas mediante la simulación, mientras que la segunda columna de cada sección muestra el ángulo real obtenido de la adquisición de datos del sistema y el procesamiento en MatLab.

Gesto	Meñique		Anular		Medio	
Cilíndrico	61°	60,62°	56°	56,39°	53°	52,98°
Punta	0°	0°	0°	0,03°	0°	0,08°
Gancho	32°	31,56°	28°	27,94°	22°	21,87°
Abierta	106°	106,4°	106°	105,9°	106°	105,6°
Palmar	0°	0°	0°	0,03°	38°	37,87°
Esférico	34°	33,51°	46°	45,99°	42°	41,87°
Lateral	0°	0°	11°	11,06°	21°	20,89°
Puño	0°	0°	0°	0°	0°	0°

Tabla 6.3: *Comparación de los ángulos de los dedos meñique, anular y medio.*

Gesto	Índice		Palma		Pulgar	
Cilíndrico	58°	57,28°	15°	14,97°	64°	63,62°
Punta	50°	49,28°	50°	50°	26°	26,11°
Gancho	30°	29,28°	110°	110,8°	90○	90,2°
Abierta	106°	105,9°	110°	110,8°	90°	90,2°
Palmar	47°	46,26°	40°	40°	32°	31,89°
Esférico	33°	32,3°	15°	14,94°	54°	53,76°
Lateral	32°	31,23°	100°	100,1°	18°	17,93°
Puño	0°	0°	0°	0°	15°	15°

Tabla 6.4: *Comparación de los ángulos de los dedos índice, palma y pulgar*

6.2.1. Análisis de resultados

Con ayuda de las tablas de comparación se puede observar que los dedos se posicionan adecuadamente en los ángulos indicados con un error máximo de 0.8°, por lo que se puede considerar que las estrategias de agarre se realizan adecuadamente.

Como parte final de la prueba, se ejecutaron los gestos de agarre con la mano robótica para sostener los distintos objetos previamente simulados. En la Figura 6.4 se observan las sujeciones realizadas por la mano robótica adaptándose adecuadamente a los objetos para los cuales se estimaron las posiciones.

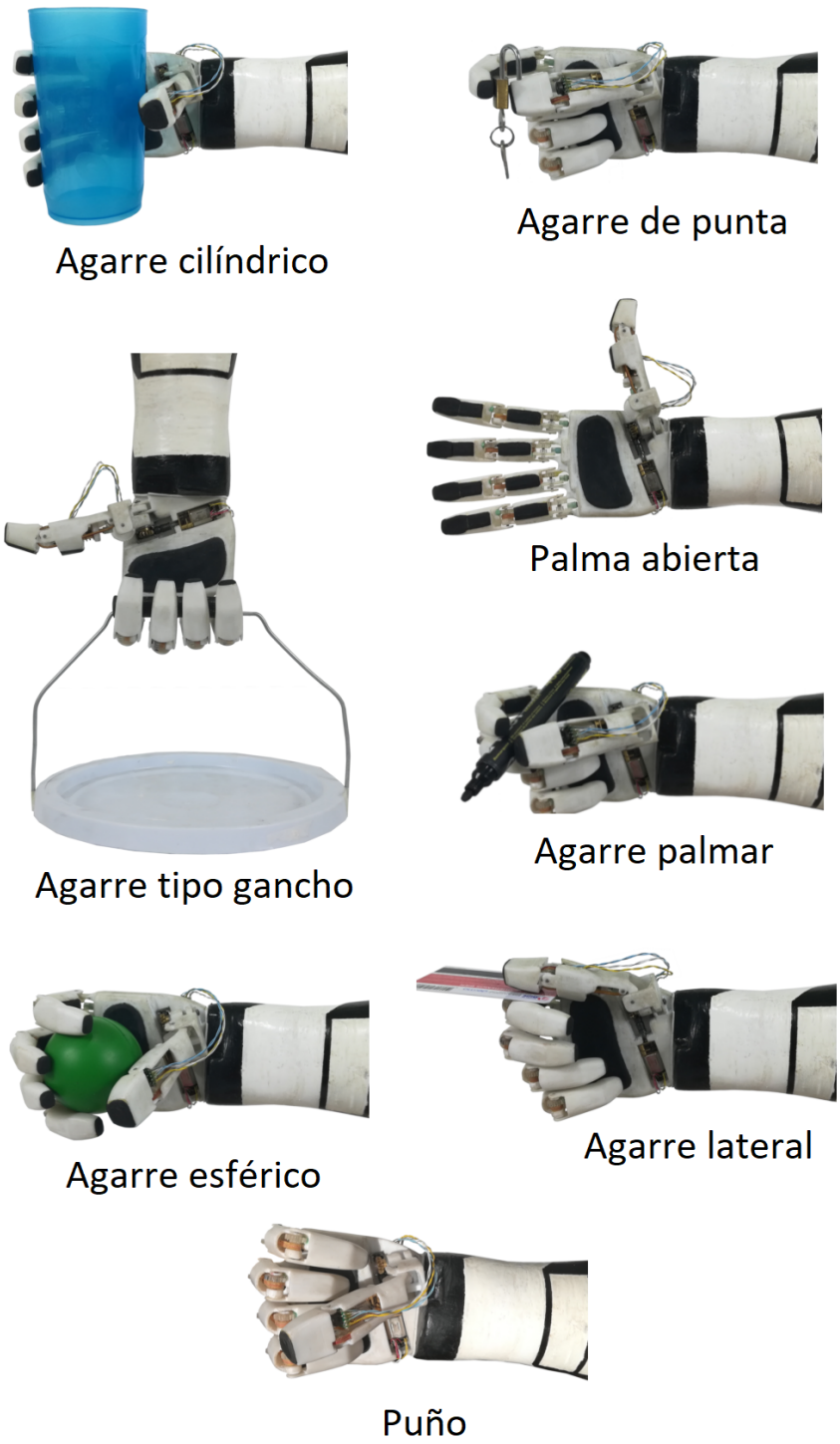


Figura 6.4: Estrategias de agarre realizadas por la mano robótica sujetando diversos objetos.

Capítulo 7

Conclusiones y trabajos futuros

En este capítulo se exponen las conclusiones obtenidas a partir del desarrollo de este trabajo. Además, se presentan algunas propuestas para el posible mejoramiento del prototipo.

7.1. Conclusiones

El desarrollo del prototipo de prótesis activa, presentado en este trabajo partió del diseño de referencia [20], al cual se le realizaron las modificaciones comentadas a continuación.

Se añadió un movimiento dependiente en la articulación de la falange distal del dedo pulgar, lo que permitió mejorar la sujeción de algunos de los objetos con los agarres lateral, palmar y esférico, ya que el dedo pulgar se adapta de mejor manera a los objetos manipulados. Se adaptaron nuevos elementos implementados en la mano robótica (motores, encoders, engranes) tomando en cuenta sus tamaños y formas. De esta manera se pudo añadir un sistema para obtener la posición de cada uno de los dedos. Se fabricó el prototipo de la mano robótica utilizando una impresora 3D y con fibra de vidrio se creó el socket de la prótesis.

La sustitución de los engranes de metal por los de acrílico redujo en gran medida el costo de adquisición y facilitó su obtención, sin embargo, los engranes de acrílico se rompen si los mecanismos se fuerzan al exceder su rango de movimiento o al atascarse con un objeto que interfiriera en la trayectoria del dedo. A pesar de tener menor resistencia, los engranes de acrílico fueron lo suficientemente resistentes para ejecutar las pruebas de control, por lo que se decidió hacer uso de ellos en el prototipo.

El uso del brazalete Myo facilitó la adquisición de señales mioeléctricas y gracias al *script* del proyecto myo-raw se obtuvieron los datos de los sensores para poder llevar a cabo la clasificación mediante la Raspberry Pi 3, sin la dependencia de una PC.

El proceso de clasificación se llevó a cabo utilizando el modelo de Memorias Asociativas Extendidas implementado en la Raspberry Pi 3 mediante una extención en leguaje C. El algoritmo clasificador es entrenado con 100 muestras de cada sensor para cada gesto a reconocer.

Una vez que el algoritmo clasifica las señales obtenidas por el brazalete y se determina el gesto a ejecutar en la mano robótica, se transmite la información al microcontrolador para que coloque cada dedo en su posición.

La Raspberry Pi 3, al requerir una fuente de alimentación de 5V (con la cual se alimenta todo el sistema) puede ser utilizada con una batería para añadir movilidad al prototipo.

Para realizar las pruebas del Capítulo 6 se desarrollaron dos programas para manipular la mano: En la primer prueba se utilizó el programa del sistema completo, que reproduce en la mano robótica los gesos indicados mediante la actividad muscular analizada con el brazalete, mientras que para la segunda prueba se programó la mano robótica para llevar a cabo una secuencia de gestos controlada por un botón para manipular los objetos sin correr el riesgo de que ejecutara un gesto erróneo y se dañara el prototipo.

A través del análisis de resultados de las pruebas de desempeño de la clasificación de datos realizadas en la Sección 6.1.1, se puede observar que el porcentaje de clasificacion satisfactorio obtenido para la primera ejecución es de 86,25 %, mientras que para la segunda se obtuvo un porcentaje del 67,5 %. Dichos resultados indican que para obtener un mejor desempeño de clasificación se debe realizar la etapa de entrenamiento cada vez que el brazalete se cambie de lugar, o bien reducir el número de gestos a clasificar, ya que como se muestra en la Tabla 6.2, el porcentaje disminuye únicamente en algunos gestos, realizando una buena clasificación (más del 80 %) en 4 de los 8 gestos.

A pesar de lo antes mencionado, se debe tomar en cuenta que las pruebas no se realizaron en una persona con amputación, por lo que el número de agarres que se puedan ejecutar dependerá del control que el usuario final tenga en sus músculos remanentes, de tal manera que en un principio se puedan ejecutar sólo algunos gestos y conforme la rehabilitación sea mayor, agregar más posiciones de agarre.

Al utilizar el diseño propuesto en [20] se logra que la mano sea capaz de realizar los gestos de agarre más comunes de la mano humana, facilitando la sujeción de objetos de distintas formas y tamaños. Sin embargo, en el prototipo desarrollado es necesario saber el ángulo al cual debe posicionarse cada dedo para sujetar adecuadamente un objeto en específico. Debido a ésto, es necesario simular cada uno de los objetos que se pretendan sujetar y estimar la posición de los dedos.

A través de un microcontrolador ATmega 328 se desarrolló un sistema para controlar los motores de la mano y generar 8 gestos de agarre predefinidos y se implementó un sistema de control de posición básico en los motores para mejorar la precisión de los movimientos. Como puede observarse en las Tablas 6.3 y 6.4 y en el análisis de resultados de la segunda prueba (Sección 6.2) el error de posición de los dedos es de 0.8°, por lo que puede considerarse que el control de posición implementado funcionó de acuerdo a las necesidades.

7.2. Trabajos futuros

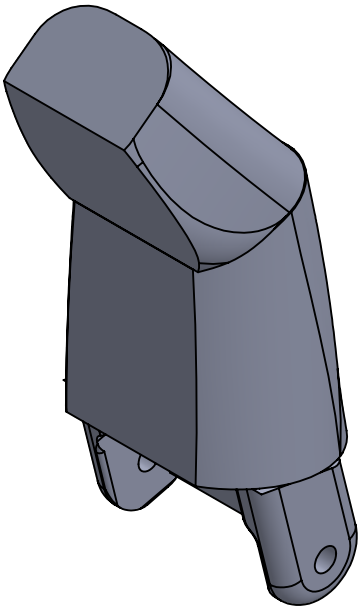
Con el fin de darle seguimiento al prototipo desarrollado en esta tesis, se realizan las siguientes propuestas de posibles puntos a mejorar.

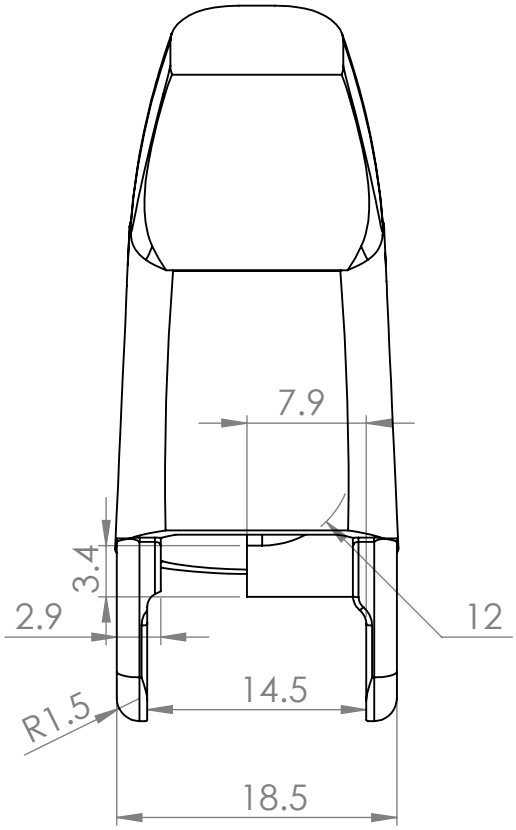
- ✓ Realizar pruebas de desempeño de clasificación obteniendo las señales EMG de los músculos remanentes en una persona con amputación de miembro superior y adecuar el número de gestos a reconocer dependiendo del control y fuerza que el usuario tenga en los músculos.

- ✓ Implementar diversos algoritmos y formas de trabajar con las señales EMG obtenidas del brazalete para mejorar el control de la mano robótica, adecuándolo más al uso del prototipo en el usuario final.
- ✓ Implementar formas alternativas para controlar la posición de los motores y mejorar su respuesta.
- ✓ Modificar el algoritmo de movimiento de los dedos de la mano robótica y agregar fuentes de alimentación para manipular varios motores simultáneamente y de esta manera ejecutar los gestos de manera más rápida y natural.
- ✓ Llevar a cabo los estudios necesarios para elegir y utilizar una batería como alimentación del sistema y hacer al prototipo más portable.
- ✓ Realizar estudios para medir la fuerza generada por la mano robótica y verificar la resistencia de los materiales que la conforman.
- ✓ Añadir un subsistema que controle la fuerza de sujección de cada uno de los dedos, ya sea utilizando el sensor de corriente integrado en los controladores de los motores o bien de algún otro sensor, con el fin de detectar la sujeción de un objeto y así evitar la necesidad de conocer la posición de cada dedo para realizar el agarre de objetos de distintos tamaños.

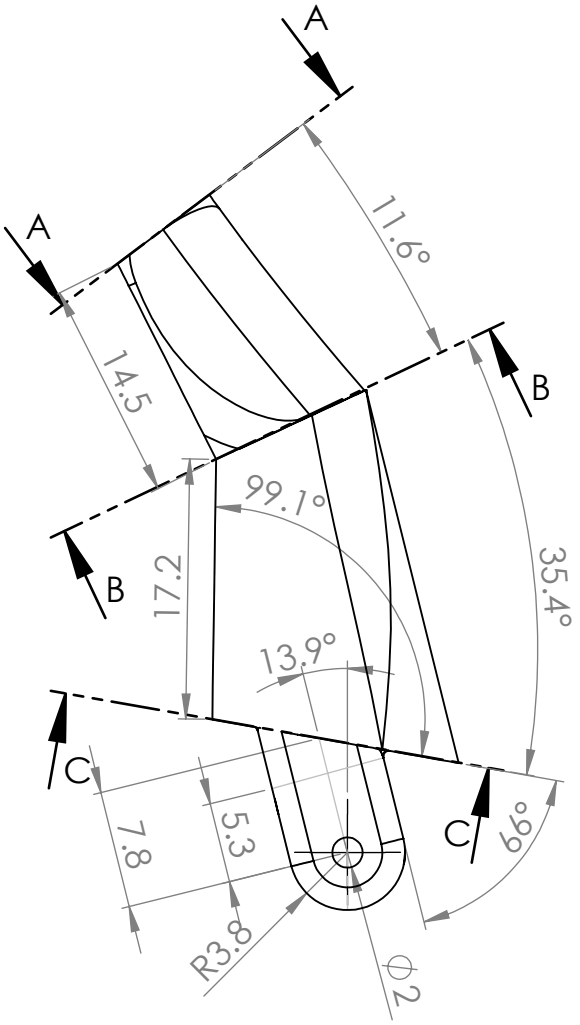
Apéndice A

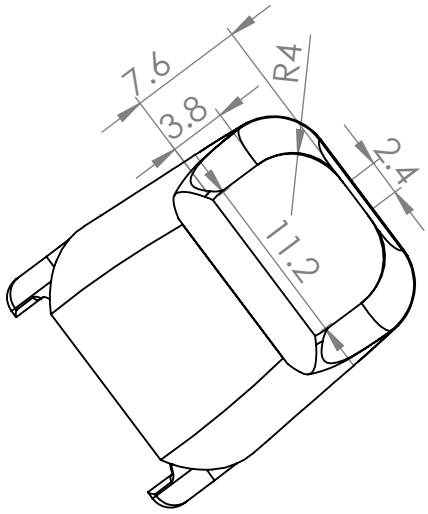
Dibujos técnicos de la mano robótica



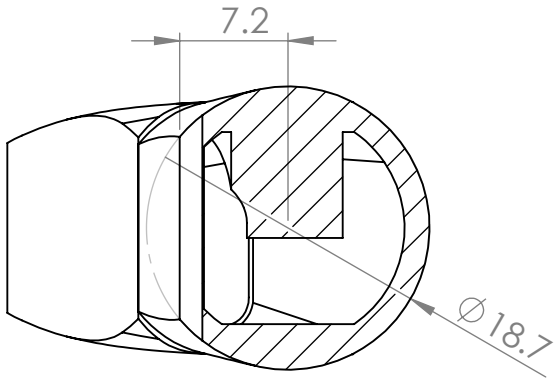


Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 4	Título: Falanges distales de los dedos opuestos al pulgar	
	Material: PLA	N.° de dibujo FDO	Carta
	Unidades: mm	Escala: 2:1	Hoja 1 de 2

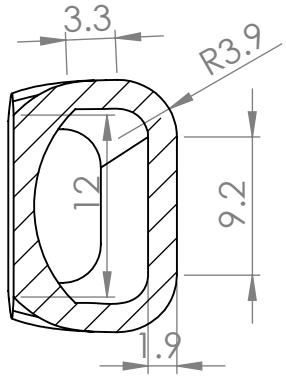




SECCIÓN A-A

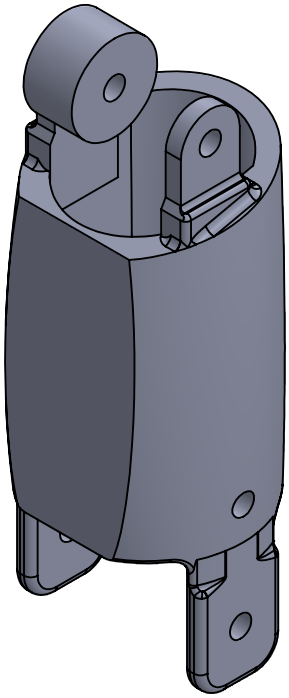


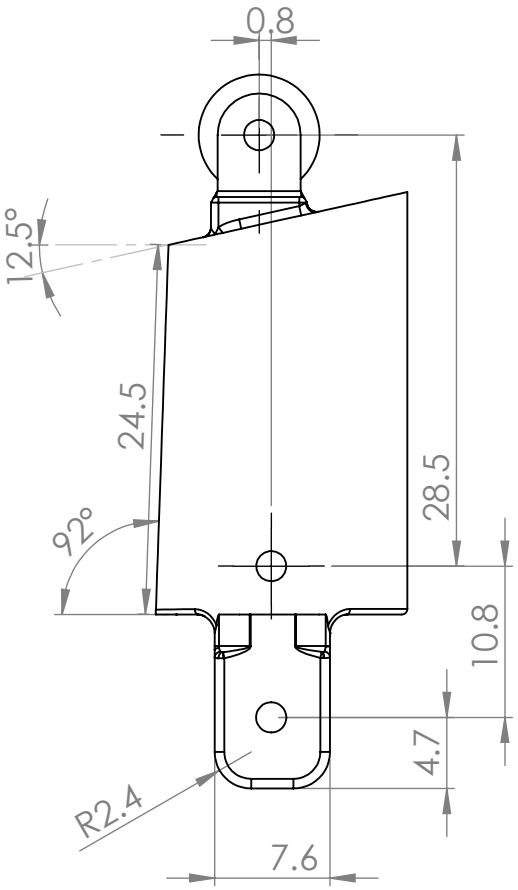
SECCIÓN C-C

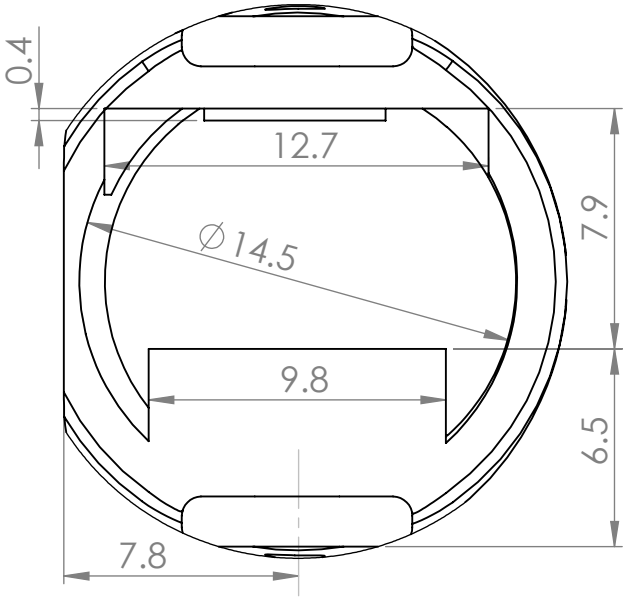


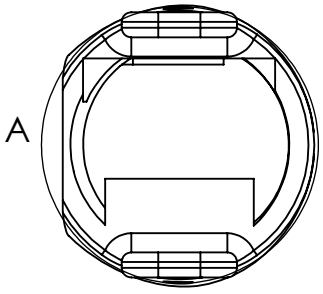
SECCIÓN B-B

Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 4	Título: Falanges distales de los dedos opuestos al pulgar	
	Material: PLA	N.° de dibujo FDO	Carta
	Unidades: mm	Escala: 2:1	Hoja 2 de 2



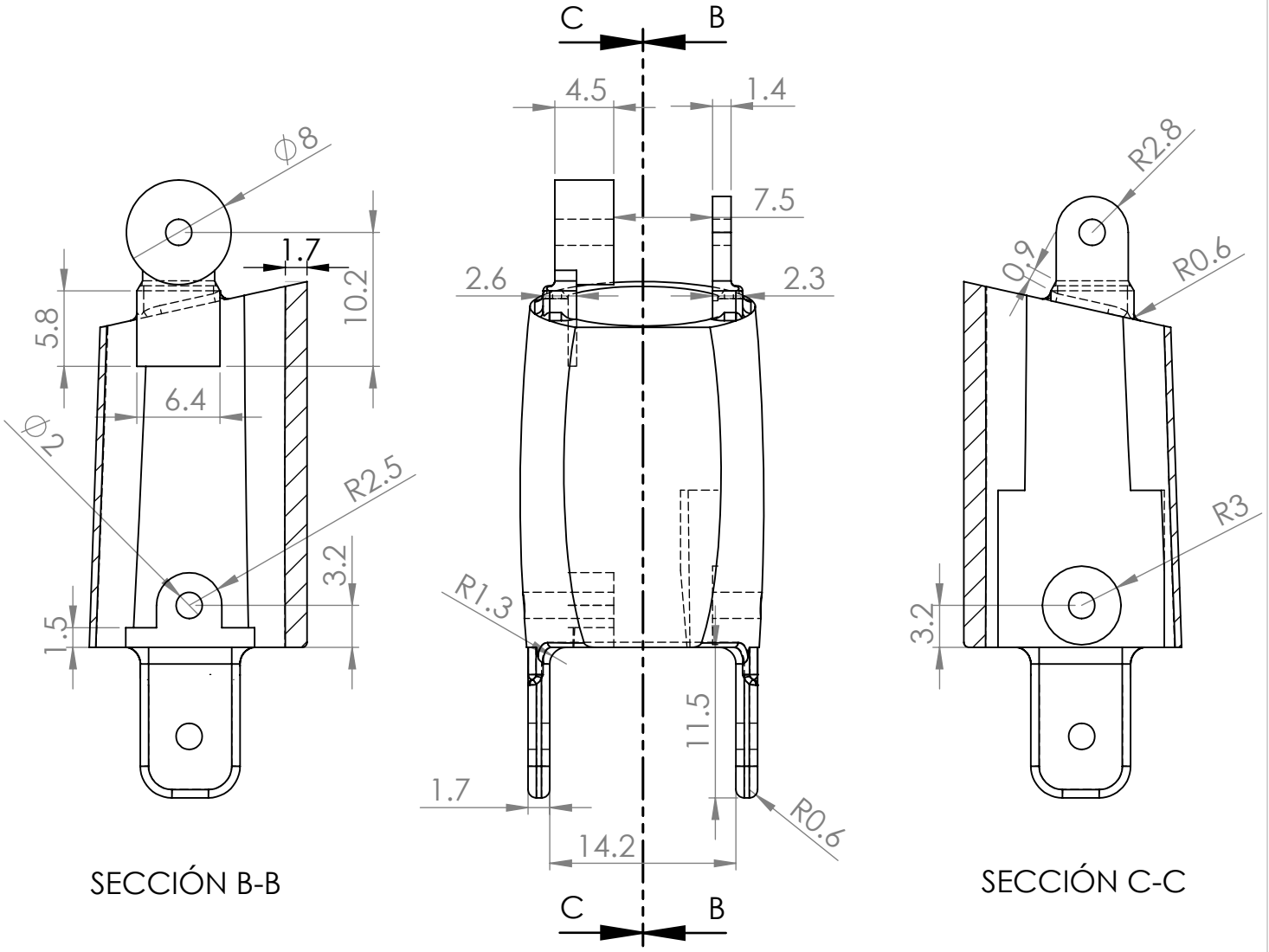






DETALLE A
ESCALA 4 : 1

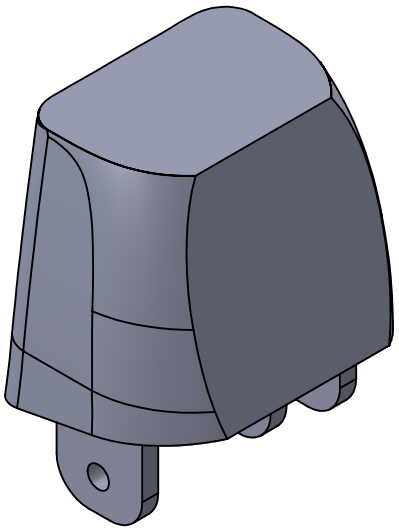
Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 4	Título: Falanges proximales de los dedos opuestos al pulgar	
	Material: PLA	N.° de dibujo FPO	Carta
	Unidades: mm	Escala: 2:1	Hoja 1 de 2

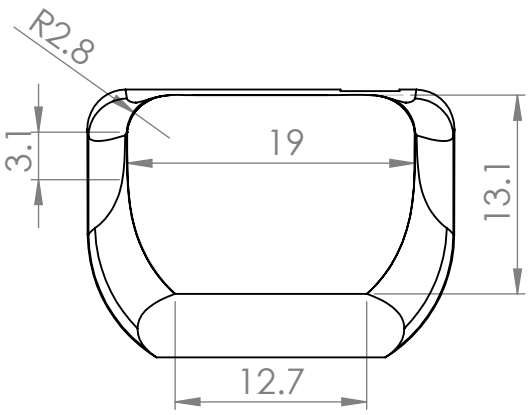


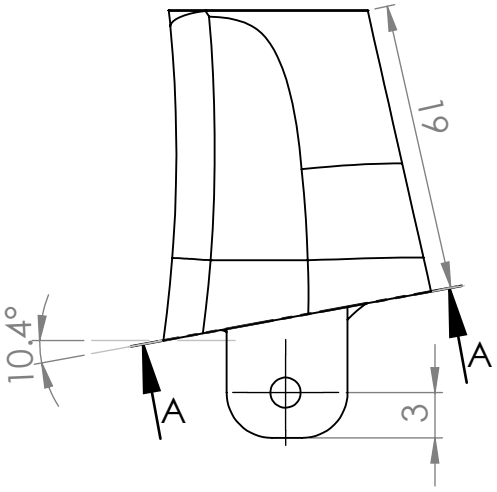
SECCIÓN B-B

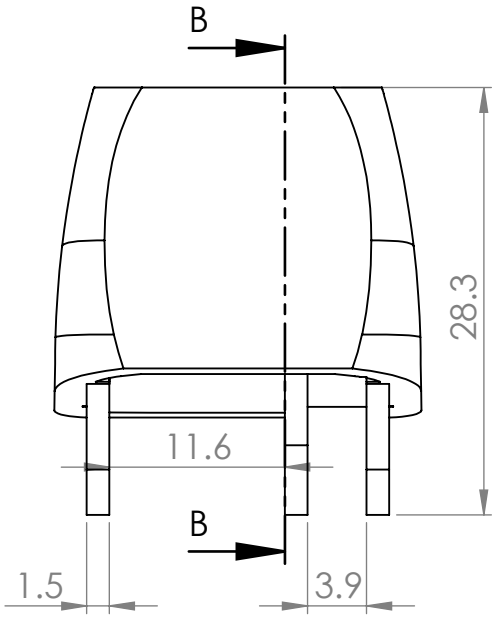
SECCIÓN C-C

Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019	
		Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 4	Título: Falanges proximales de los dedos opuestos al pulgar	
	Material: PLA	N.° de dibujo FPO	Carta
	Unidades: mm	Escala: 2:1	Hoja 2 de 2

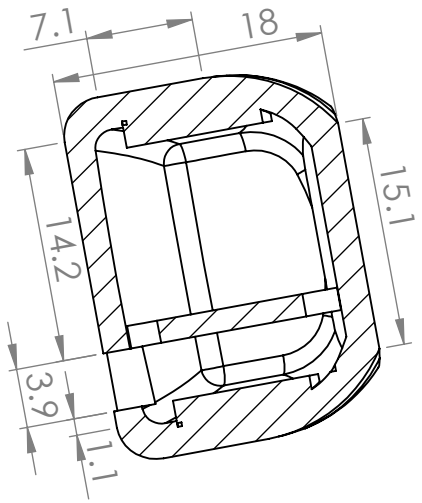




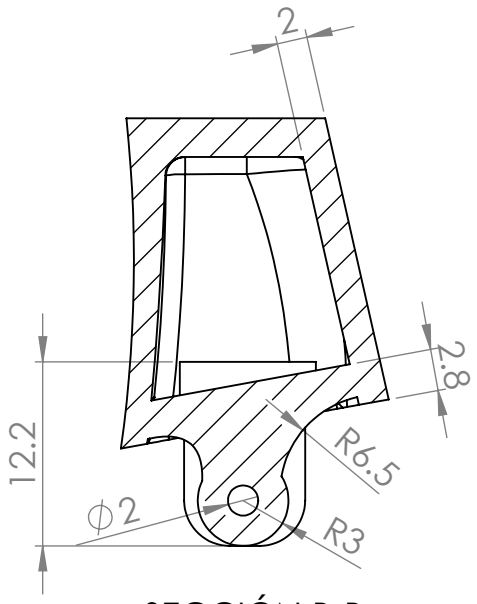




Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 1	Título: Falange distal del dedo pulgar	
	Material: PLA	N.° de dibujo FDP	Carta
	Unidades: mm	Escala: 2:1	Hoja 1 de 2

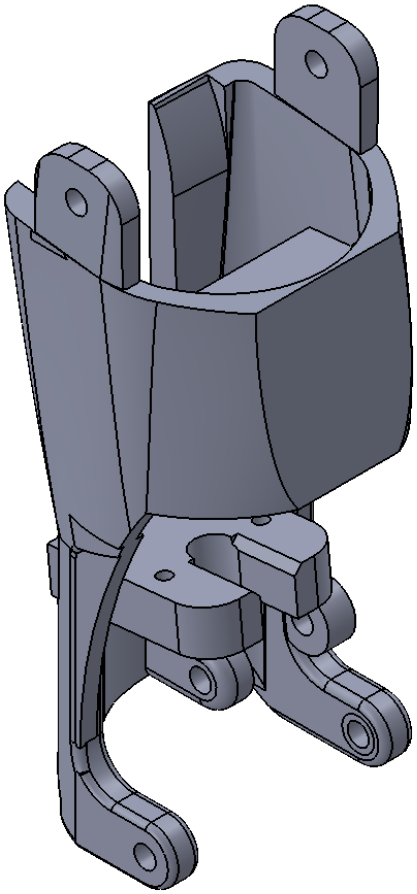


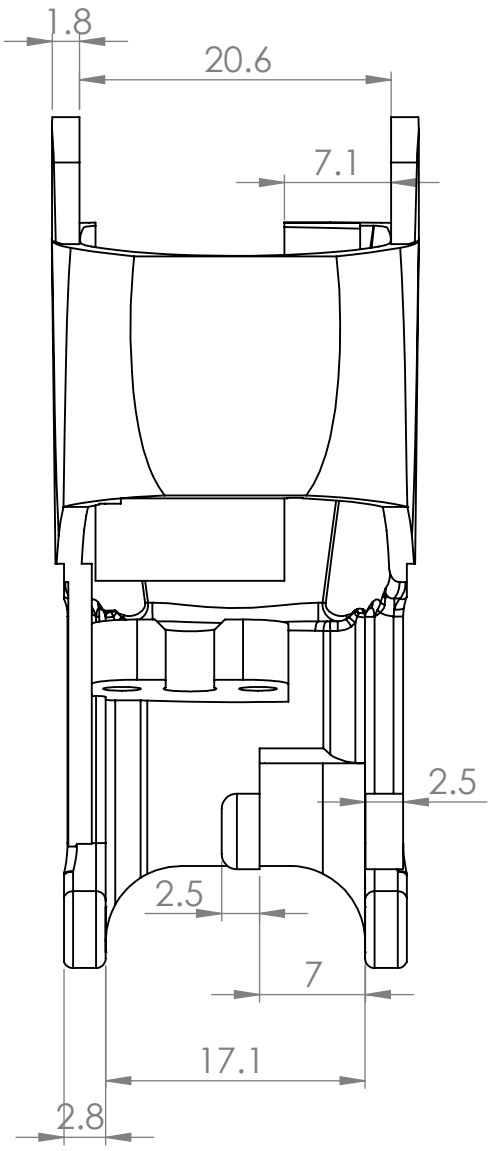
SECCIÓN A-A



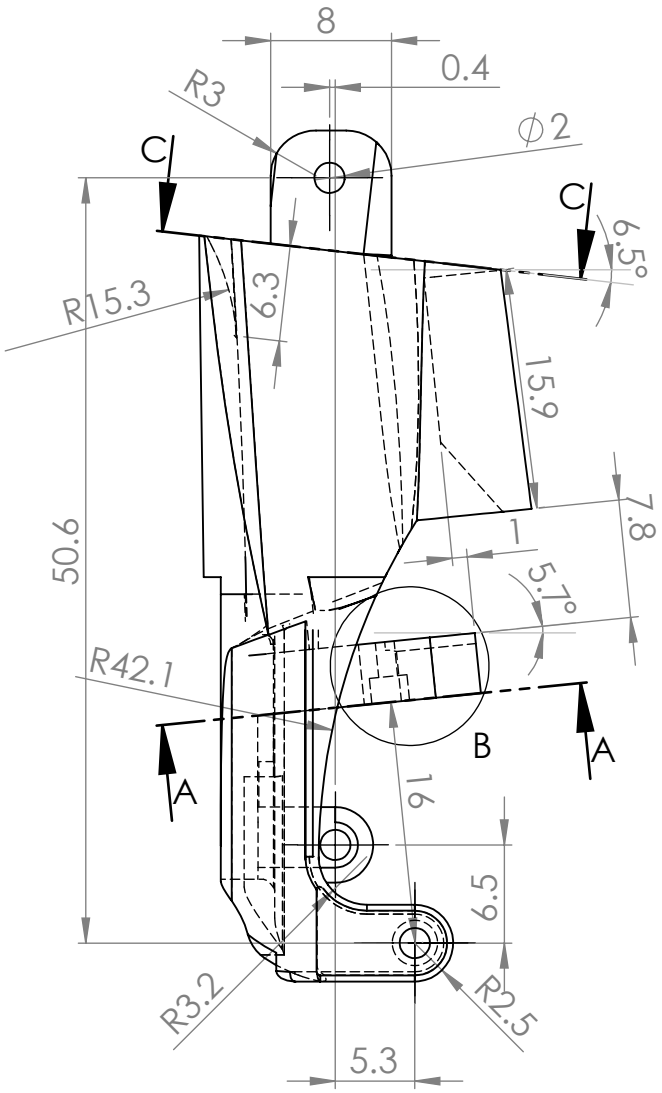
SECCIÓN B-B

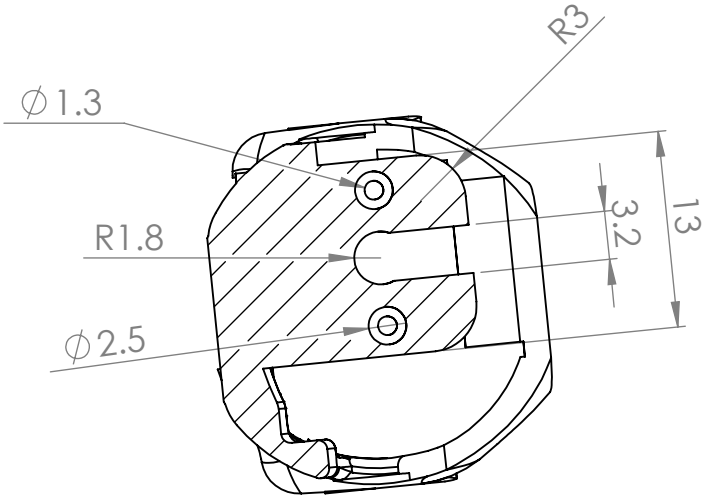
Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 1	Título: Falange distal del dedo pulgar	
	Material: PLA	N.° de dibujo FDP	Carta
	Unidades: mm	Escala: 2:1	Hoja 2 de 2



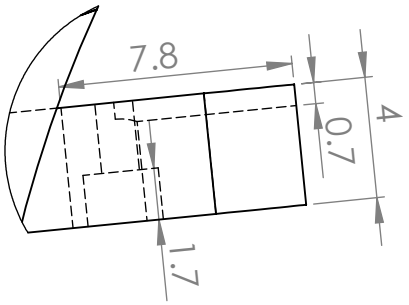


Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 1	Título: Falange proximal del dedo pulgar	
	Material: PLA	N.° de dibujo FPP	Carta
	Unidades: mm	Escala: 2:1	Hoja 1 de 3



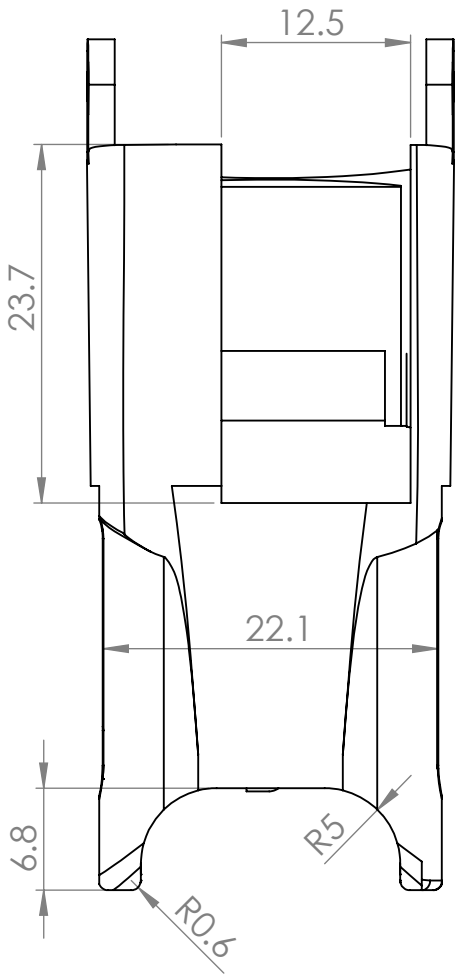


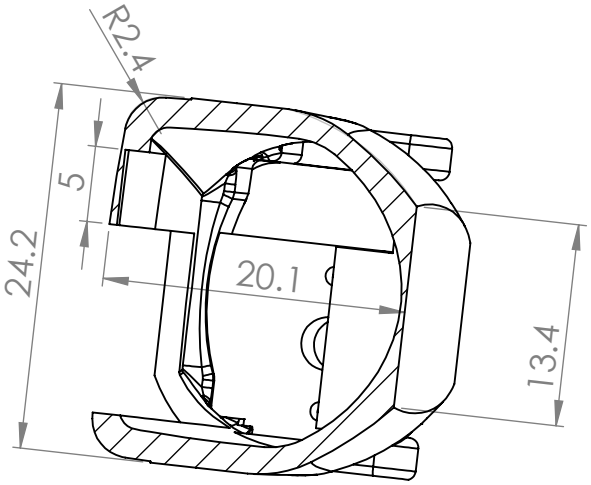
SECCIÓN A-A
ESCALA 2 : 1



DETALLE B
ESCALA 4 : 1

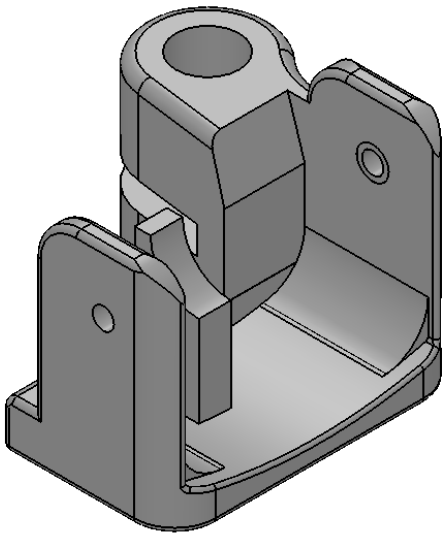
Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 1	Título: Falange proximal del dedo pulgar	
	Material: PLA	N.° de dibujo FPP	Carta
	Unidades: mm	Escala: 2:1	Hoja 2 de 3

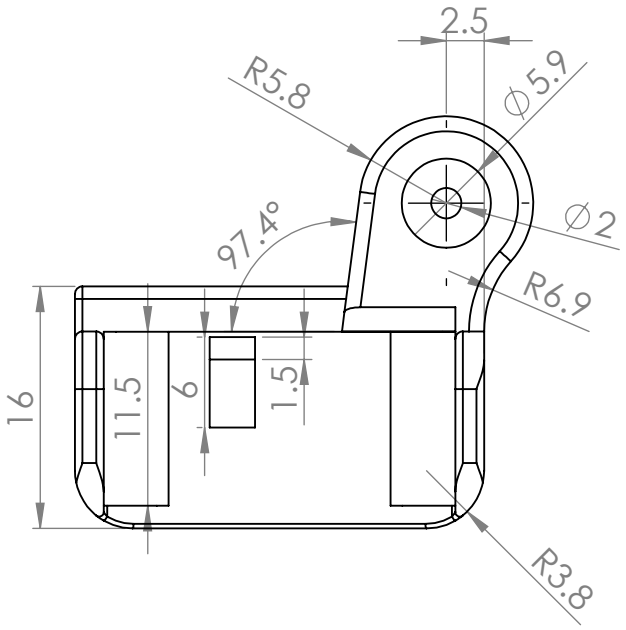


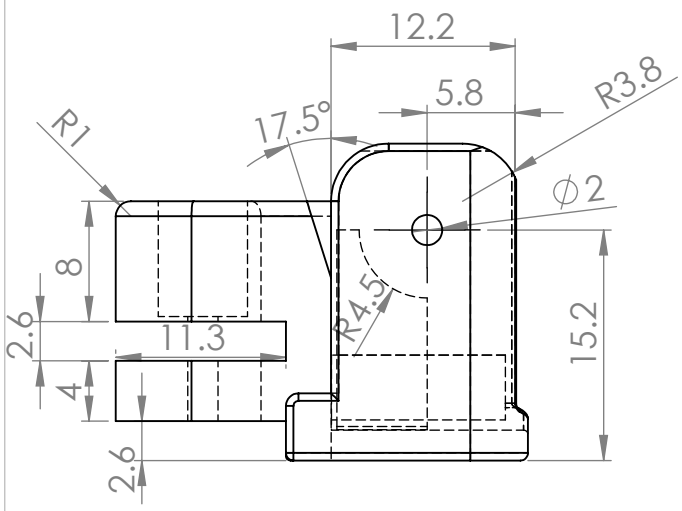


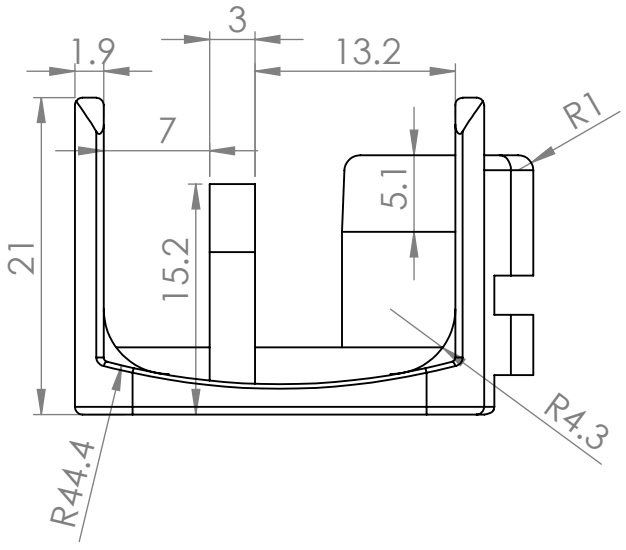
SECCIÓN C-C
ESCALA 2 : 1

Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019	
		Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 1	Título: Falange proximal del dedo pulgar	
	Material: PLA	N.° de dibujo FPP	Carta
	Unidades: mm	Escala: 2:1	Hoja 3 de 3

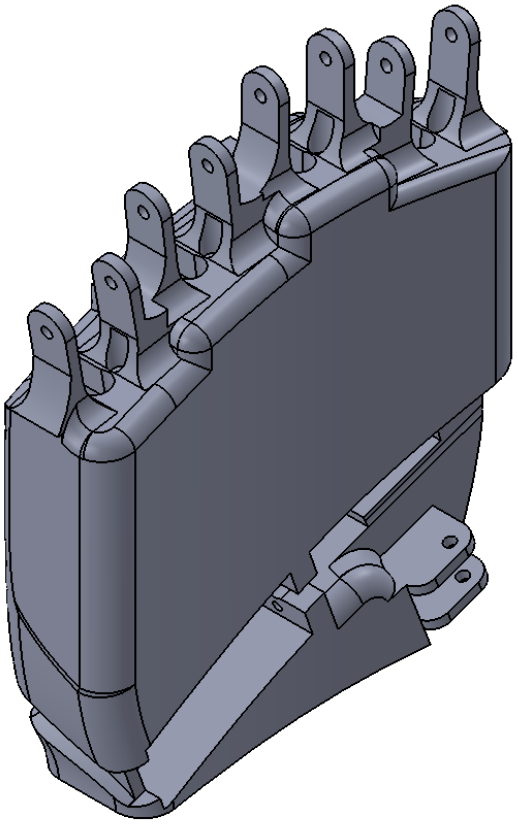


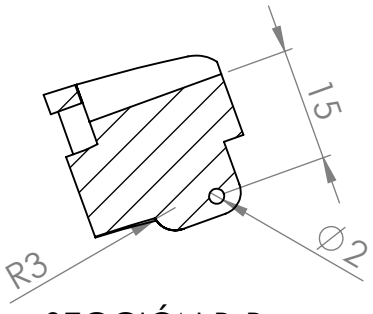




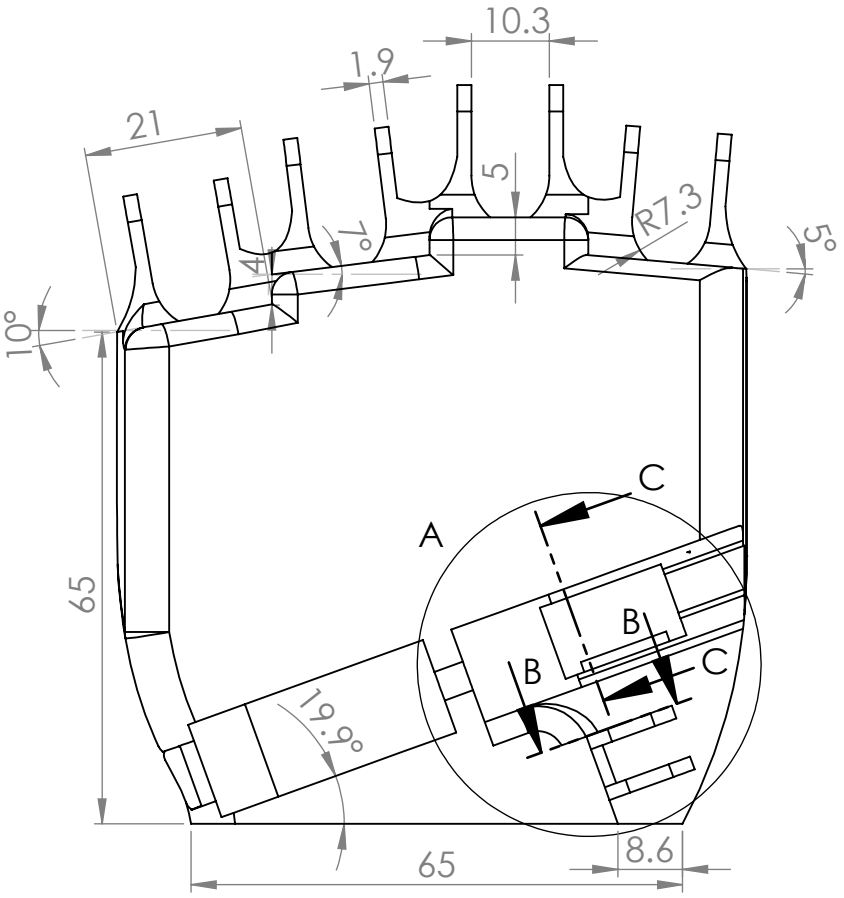


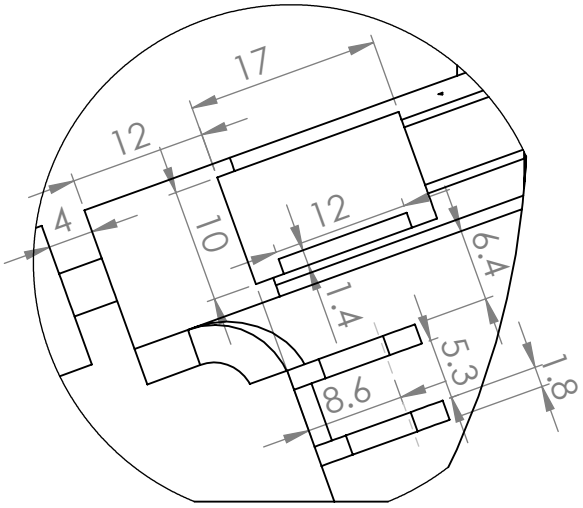
Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 1	Título: Base del dedo pulgar	
	Material: PLA	N.° de dibujo BDP	Carta
	Unidades: mm	Escala: 2:1	Hoja 1 de 1





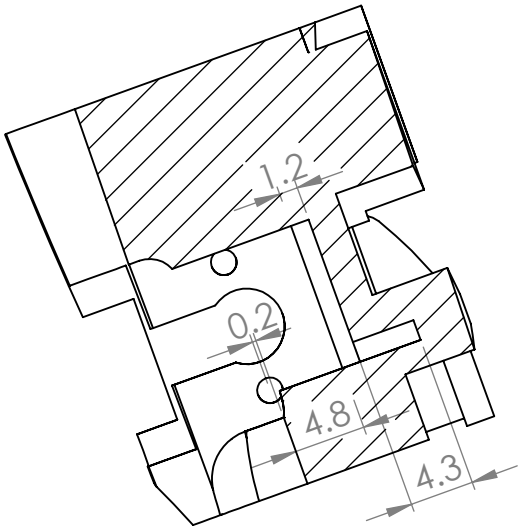
SECCIÓN B-B



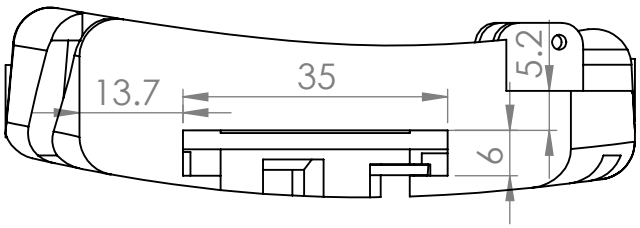


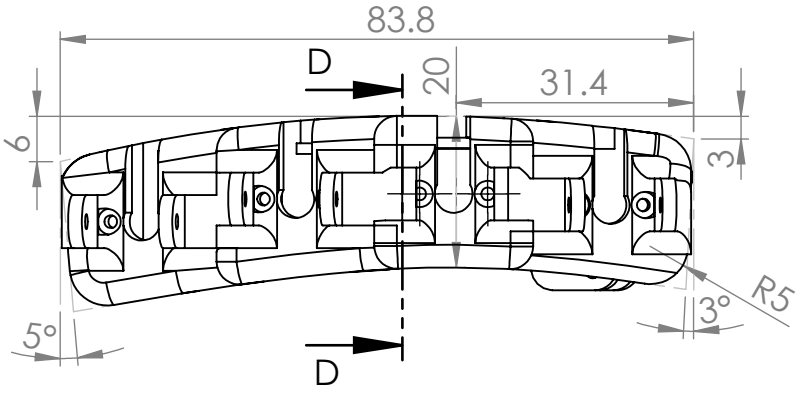
DETALLE A
ESCALA 1.5 : 1

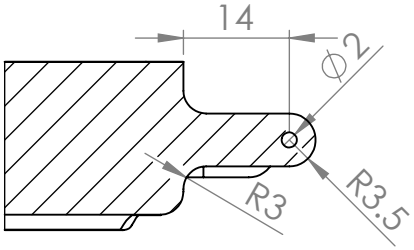
Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 1	Título: Palma de la mano	
	Material: PLA	N.° de dibujo PM	Carta
	Unidades: mm	Escala: 1:1	Hoja 1 de 3



SECCIÓN C-C
ESCALA 2 : 1

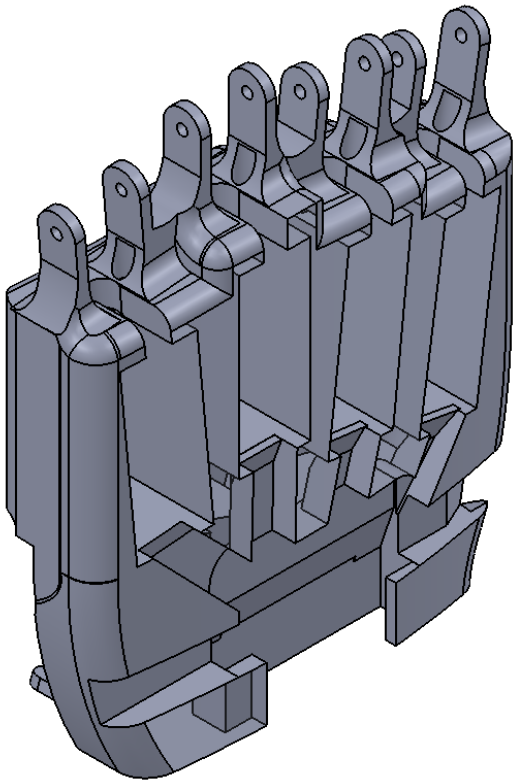


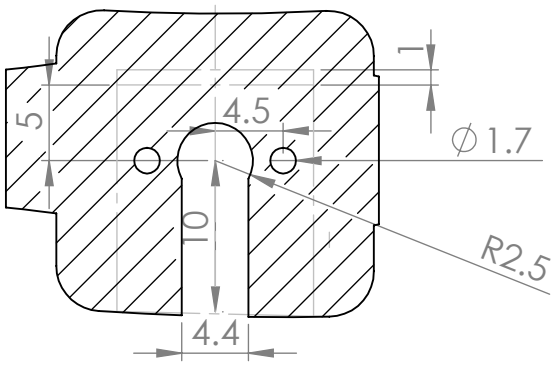




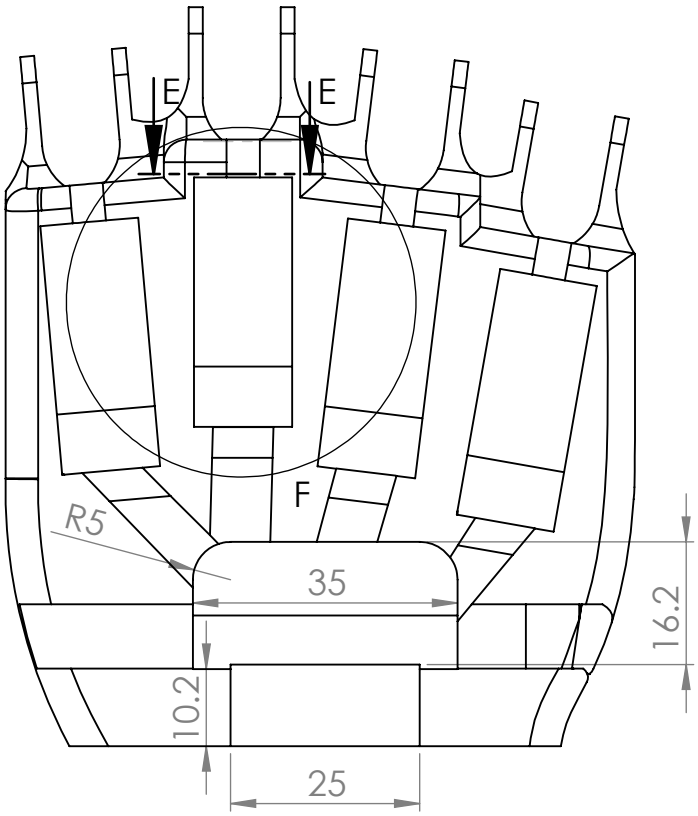
SECCIÓN D-D

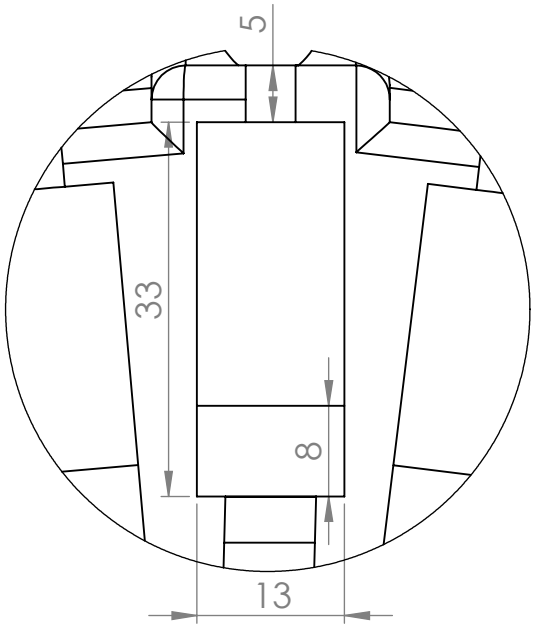
Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 1	Título: Palma de la mano	
	Material: PLA	N.° de dibujo PM	Carta
	Unidades: mm	Escala: 1:1	Hoja 2 de 3





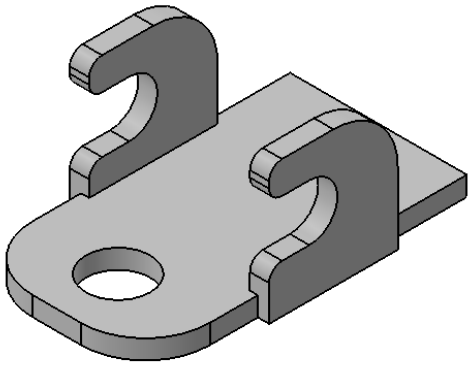
SECCIÓN E-E
ESCALA 2 : 1

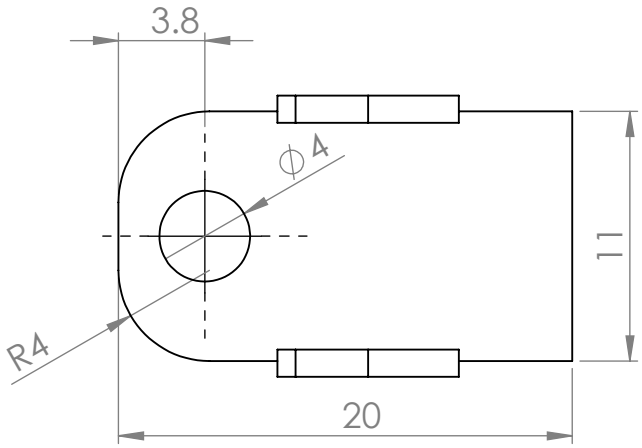


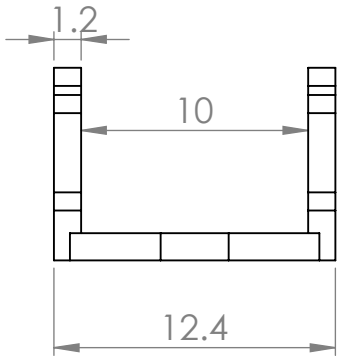


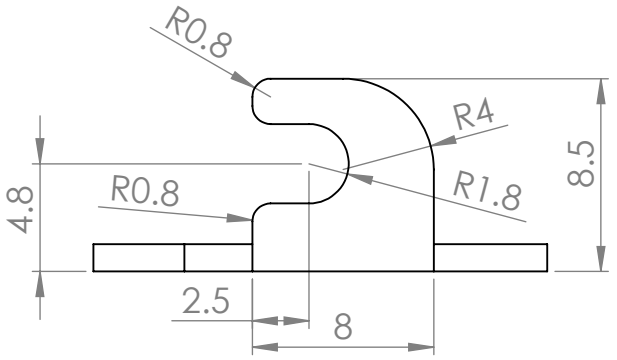
DETALLE F
ESCALA 1.5 : 1

Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 1	Título: Palma de la mano	
	Material: PLA	N.° de dibujo PM	Carta
	Unidades: mm	Escala: 1:1	Hoja 3 de 3

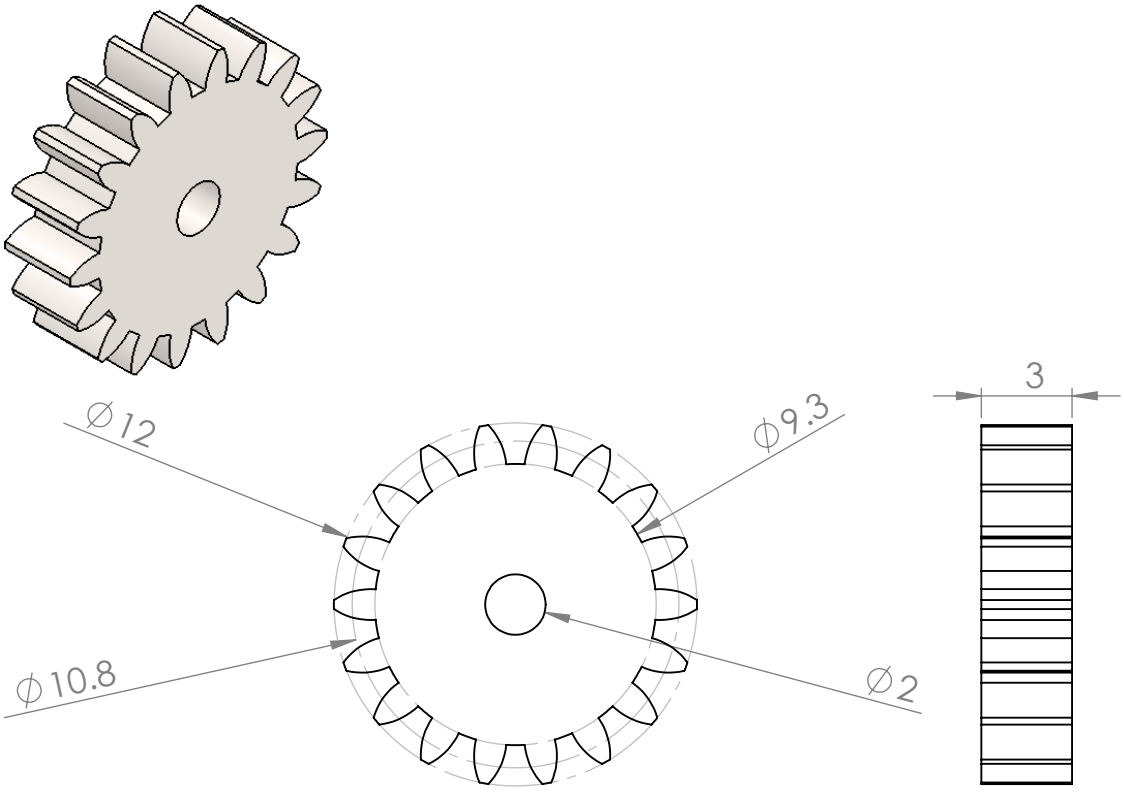






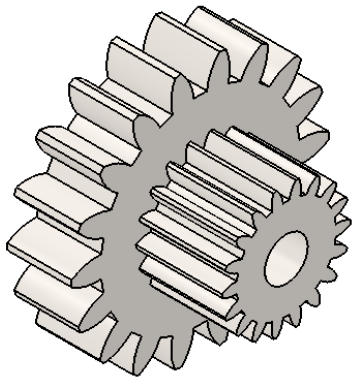


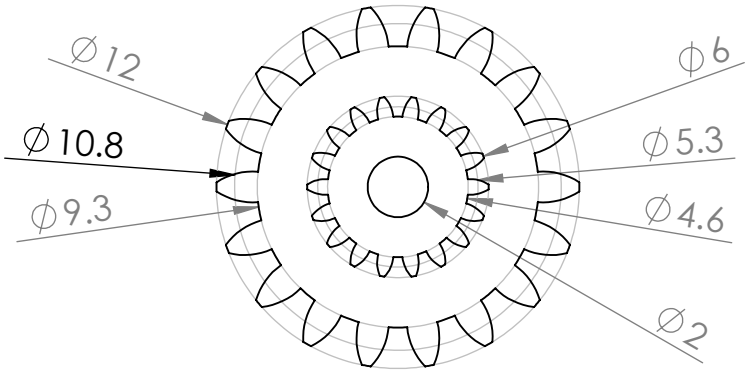
Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019	
		Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 1	Título: Soporte para tornillo sinfin	
	Material: Latón	N.° de dibujo STS	Carta
	Unidades: mm	Escala: 3:1	Hoja 1 de 1

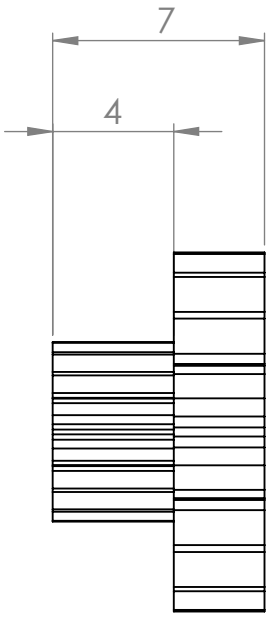


No. de dientes: 18
Módulo: 0.6

Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 4	Título: Engrane de 18 dientes	
	Material: Acrílico	N.° de dibujo E18	Carta
	Unidades: mm	Escala: 4:1	Hoja 1 de 1



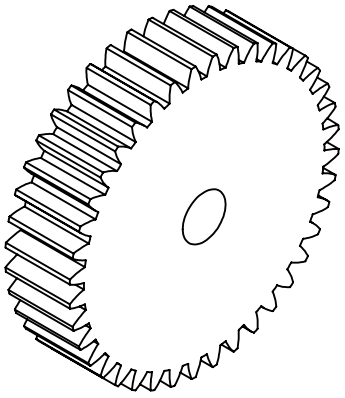


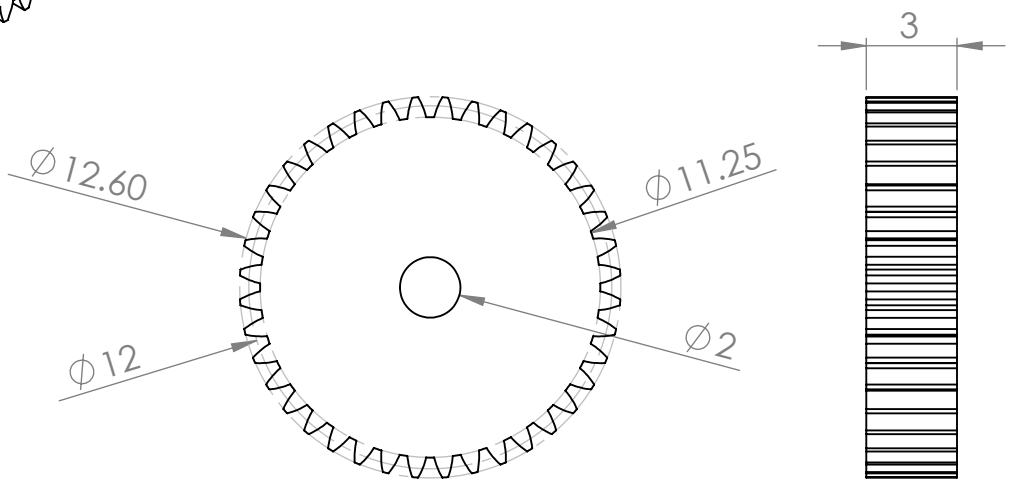


Engrane
No. de dientes: 18
Módulo: 0.6

Polea dentada
No. de dientes: 18
Módulo:0.3

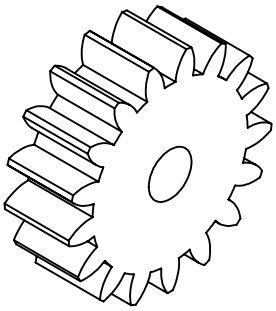
Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 8	Título: Engrane con polea de 18-18 dientes	
	Material: Acrílico	N.° de dibujo EP1818	Carta
	Unidades: mm	Escala: 3:1	Hoja 1 de 1

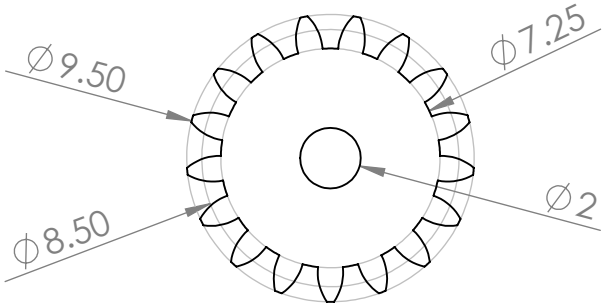


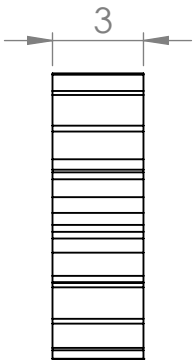


No. de dientes: 40
Módulo: 0.3

Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 1	Título: Polea de 40 dientes	
	Material: Acrílico	N.° de dibujo P40	Carta
	Unidades: mm	Escala: 4:1	Hoja 1 de 1

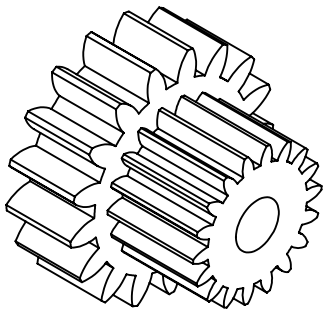


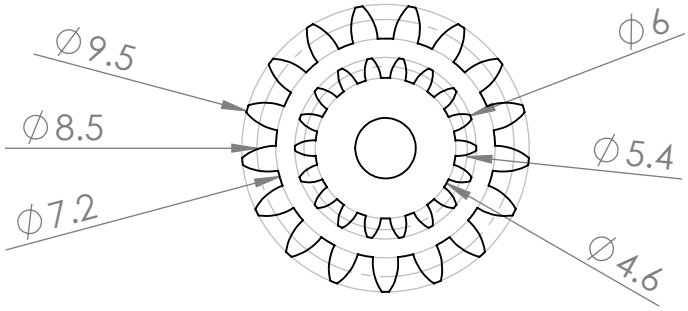


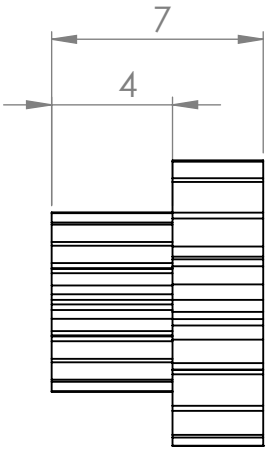


No. de dientes: 17
Módulo: 0.5

Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 1	Título: Engrane de 17 dientes	
	Material: Acrílico	N.° de dibujo E17	Carta
	Unidades: mm	Escala: 4:1	Hoja 1 de 1



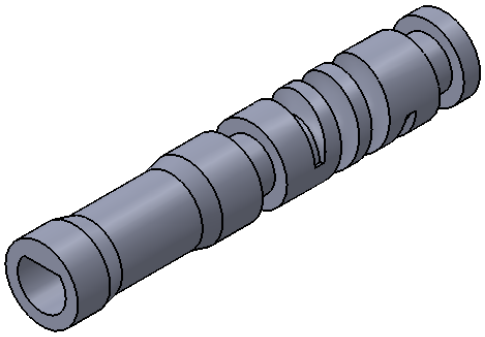


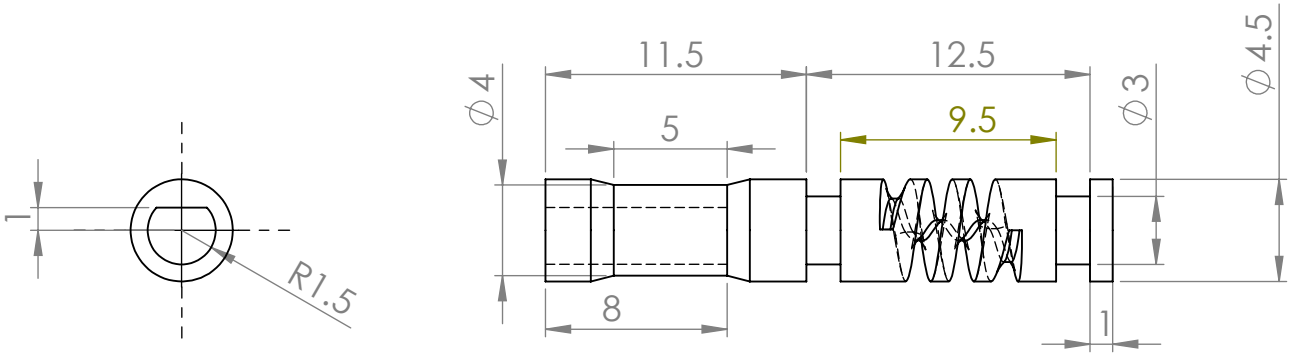


Engrane
No. de dientes: 17
Módulo: 0.5

Polea dentada
No. de dientes: 18
Módulo: 0.3

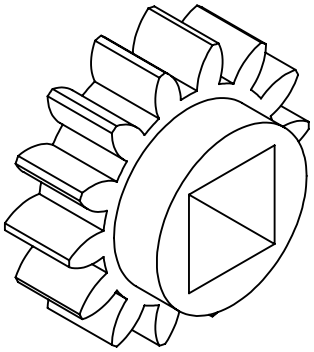
Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 1	Título: Engrane con polea de 17-18 dientes	
	Material: Acrílico	N.° de dibujo EP1718	Carta
	Unidades: mm	Escala: 4:1	Hoja 1 de 1

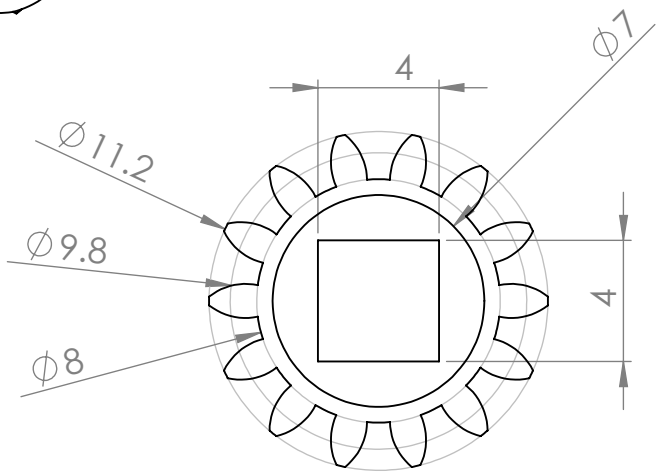


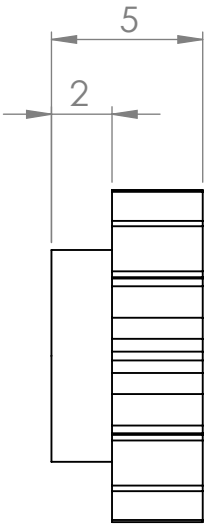


Paso de rosca: 0.2
No. de revoluciones: 3

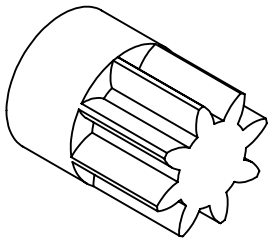
Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 1	Título: Tornillo Sinfín	
	Material: Acero	N.° de dibujo TSF	Carta
	Unidades: mm	Escala: 3:1	Hoja 1 de 1

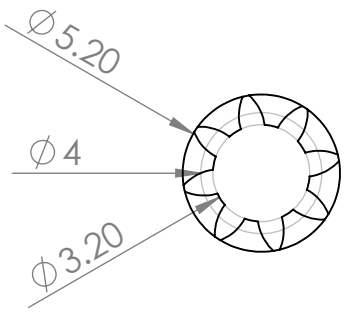


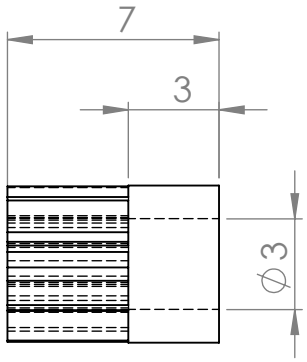




Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 1	Título: Corona de 14 dientes	
	Material: Latón	N.° de dibujo C14	Carta
	Unidades: mm	Escala: 4:1	Hoja 1 de 1

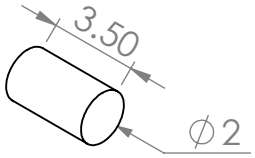




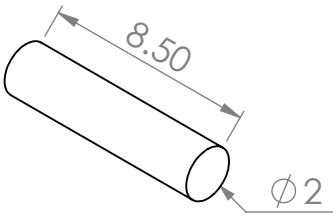


No. de dientes: 8
Módulo: 0.5

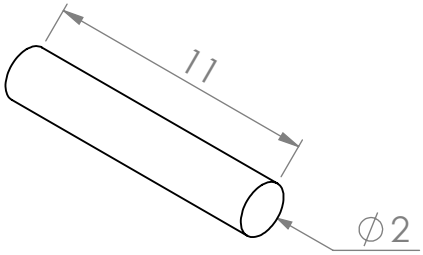
Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 5	Título: Engrane de 8 dientes	
	Material: Acrílico	N.° de dibujo E8	Carta
	Unidades: mm	Escala: 4:1	Hoja 1 de 1



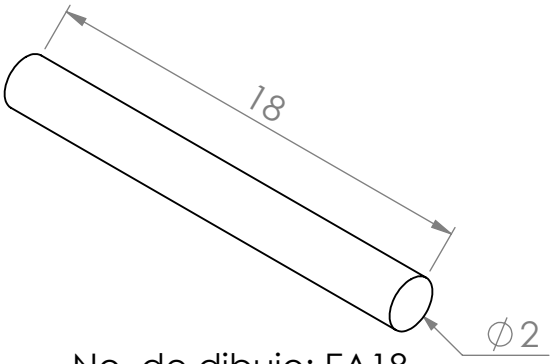
No. de dibujo: EA3
Cantidad: 1



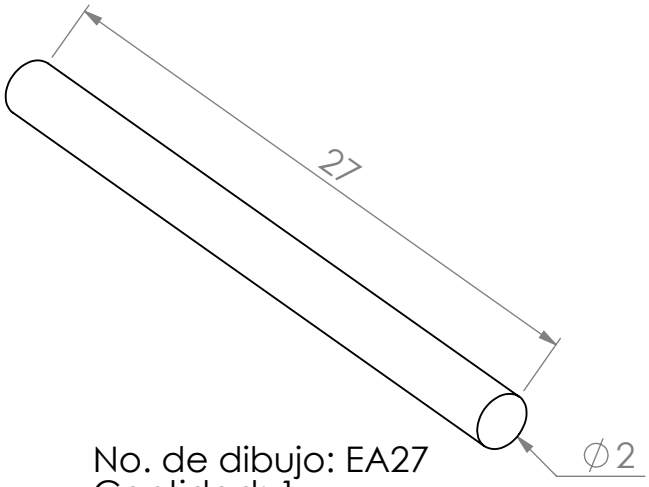
No. de dibujo: EA8
Cantidad: 1



No. de dibujo: EA11
Cantidad: 1

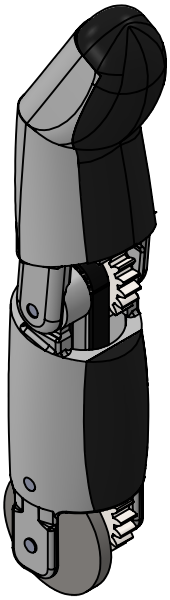


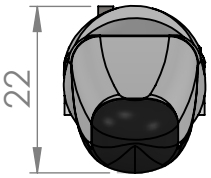
No. de dibujo: EA18
Cantidad: 12

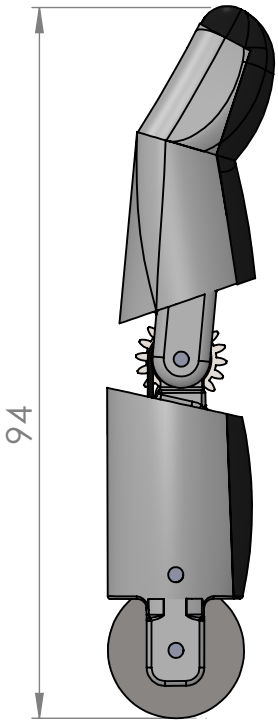


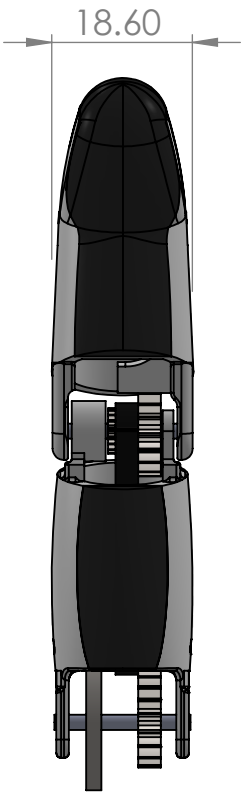
No. de dibujo: EA27
Cantidad: 1

Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: -	Título: Ejes de articulaciones	
	Material: Acero	N.° de dibujo -	Carta
	Unidades: mm	Escala: 4:1	Hoja 1 de 1

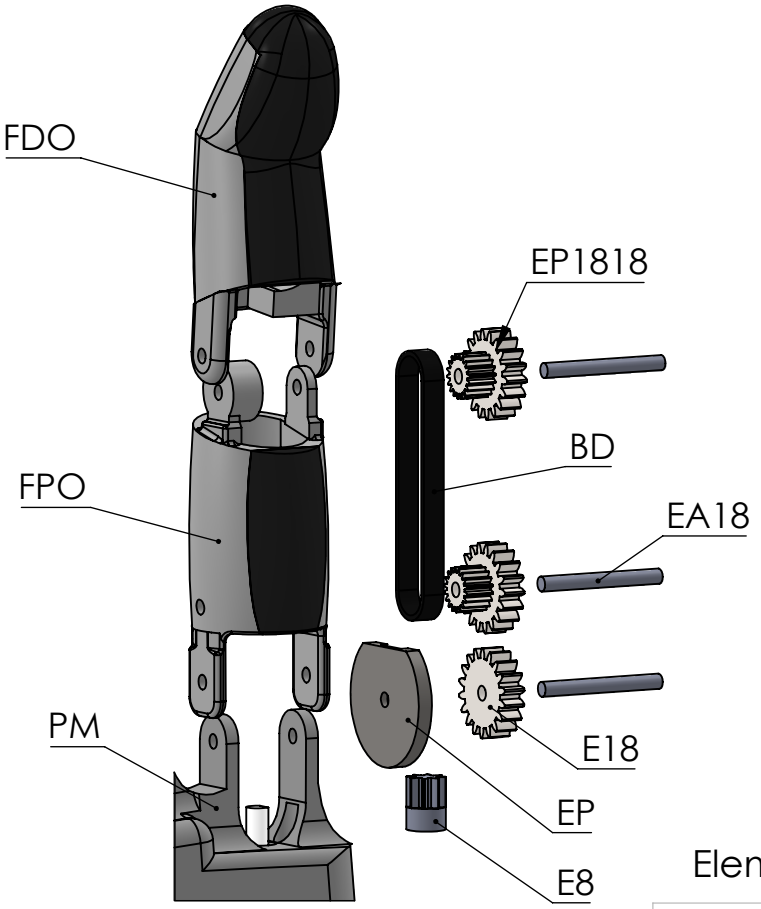








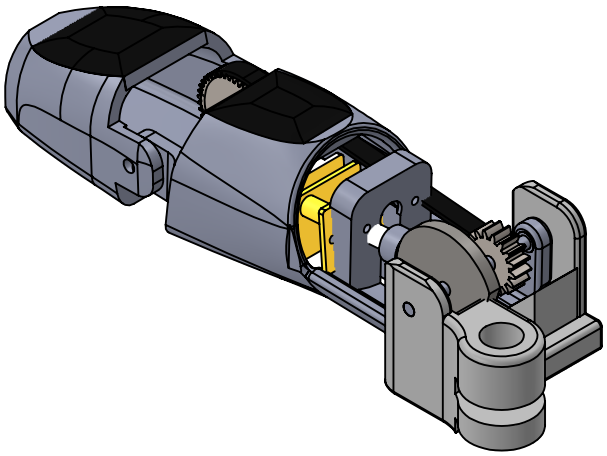
Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 4	Título: Ensamblaje de un dedo opuesto al pulgar	
	Material: -	N.° de dibujo ENSD	Carta
	Unidades: mm	Escala: 1:1	Hoja 1 de 2

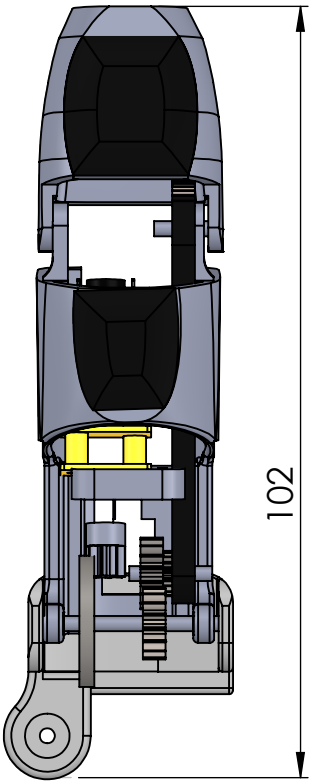


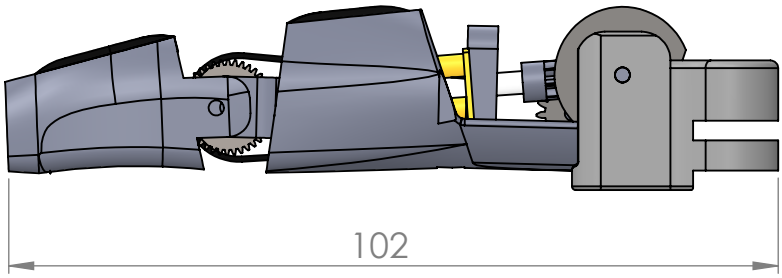
Elementos de un dedo opuesto al pulgar

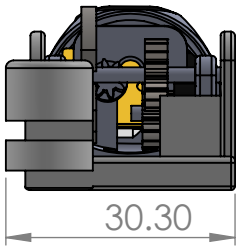
No. de dibujo	Título	Cant.
FDO	Falanges distales de los dedos opuestos al pulgar	1
FPO	Falanges proximales de los dedos opuestos al pulgar	1
PM	Palma de la mano	1
EP1818	Engrane con polea de 18-18 dientes	2
BD	Banda dentada	1
EA18	Eje de articulación de 18mm	3
E18	Engrane de 18 dientes	1
EP	Engrane perpendicular	1
E8	Engrane de 8 dientes	1

Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 4	Título: Ensamblaje explosionado de un dedo opuesto al pulgar	
	Material: -	N.° de dibujo EXPD	Carta
	Unidades: mm	Escala: 1:1	Hoja 2 de 2

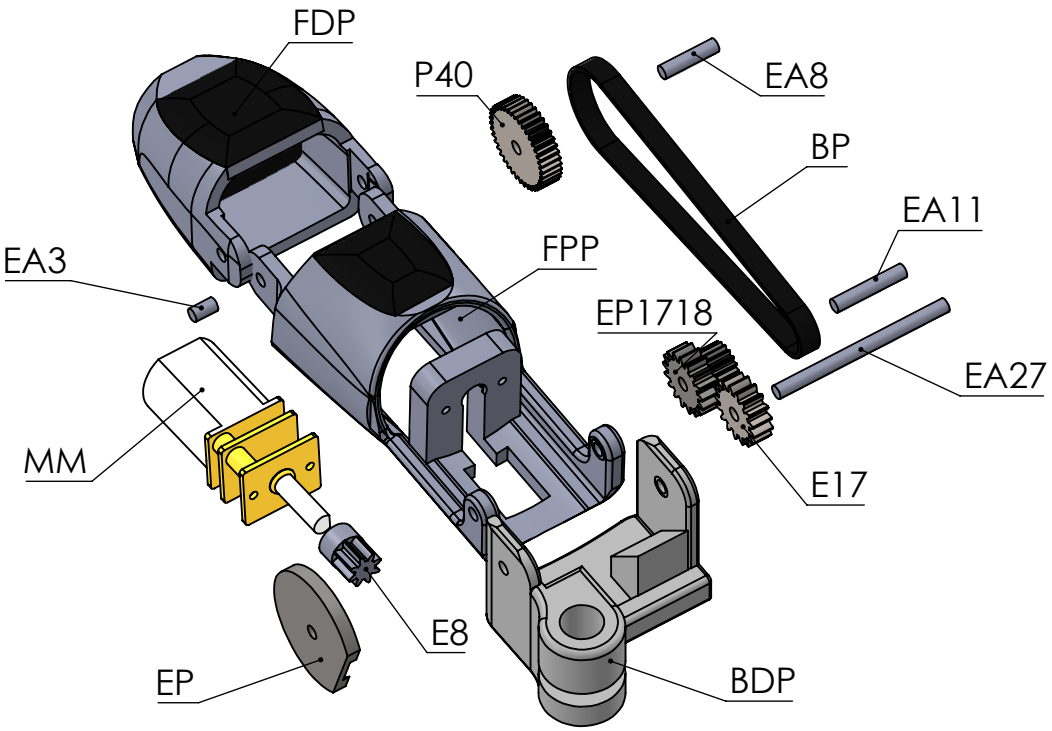








Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 1	Título: Ensamblaje del dedo pulgar	
	Material: -	N.° de dibujo ENSP	Carta
	Unidades: mm	Escala: 1:1	Hoja 1 de 2



No. de dibujo	Título	Cant.
FDP	Falange distal del dedo pulgar	1
FPP	Falange proximal del dedo pulgar	1
MM	Micromotor	1
EP	Engrane perpendicular	1
BDP	Base del dedo pulgar	1
E8	Engrane de 8 dientes	1
EA3	Eje de articulación de 3.5mm	1
P40	Polea de 40 dientes	1
EP1718	Engrane con polea de 17-18 dientes	1
E17	Engrane de 17 dientes	1
BP	Banda dentada del dedo pulgar	1
EA8	Eje de articulación de 8.5mm	1
EA11	Eje de articulación de 11mm	1
EA11	Eje de articulación de 27mm	1

Tesis:
Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas

Enero 2019

Rediseñado por:
Leobardo Elí Sánchez Velasco

Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de:

0.2 mm

Cantidad: 1

Título:
Ensamblaje explosionado del dedo pulgar

Material:
-

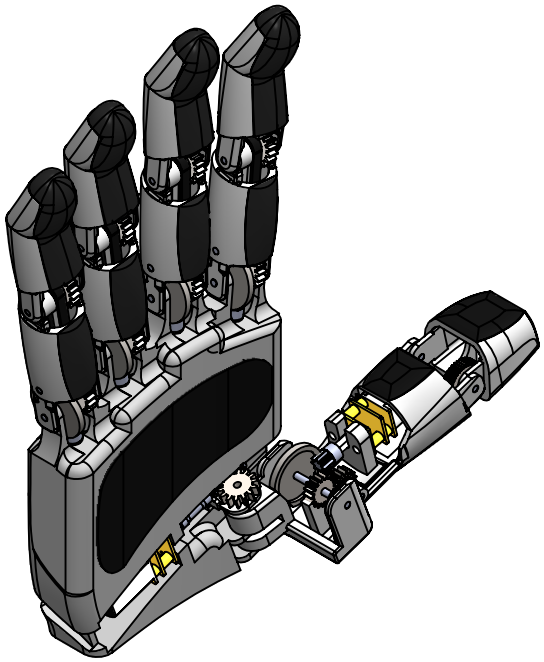
N.° de dibujo
EXPP

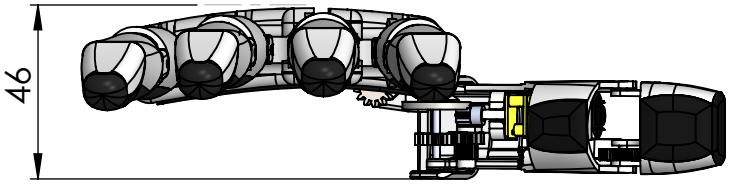
Carta

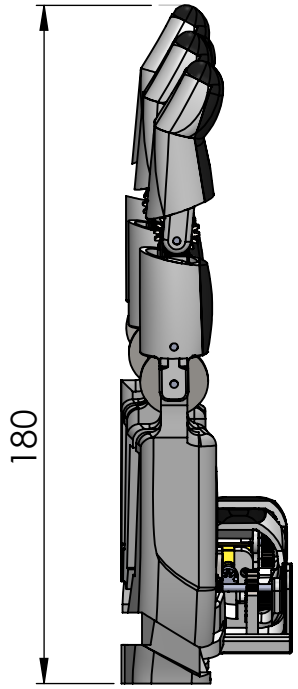
Unidades: mm

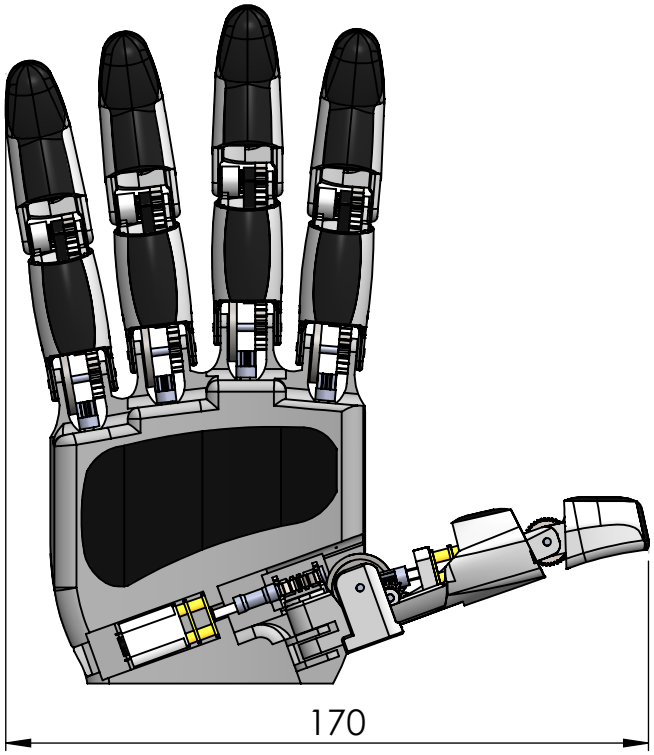
Escala: 1:1

Hoja 2 de 2

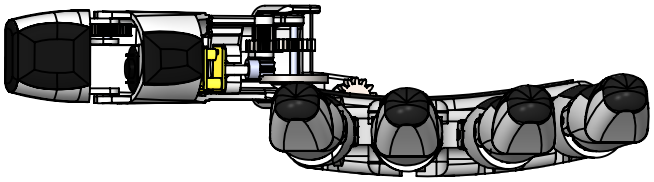


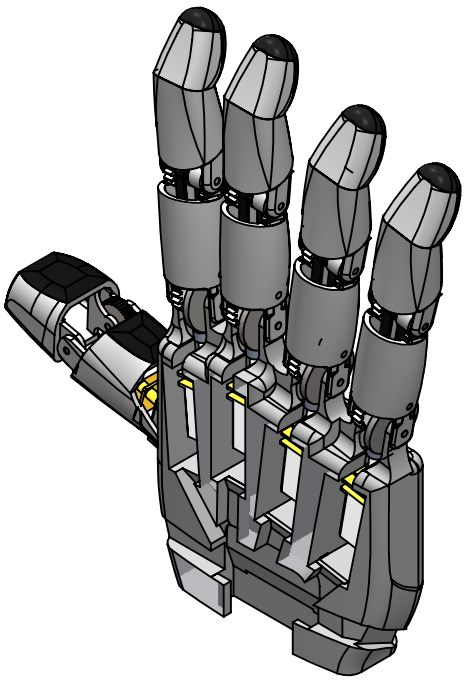


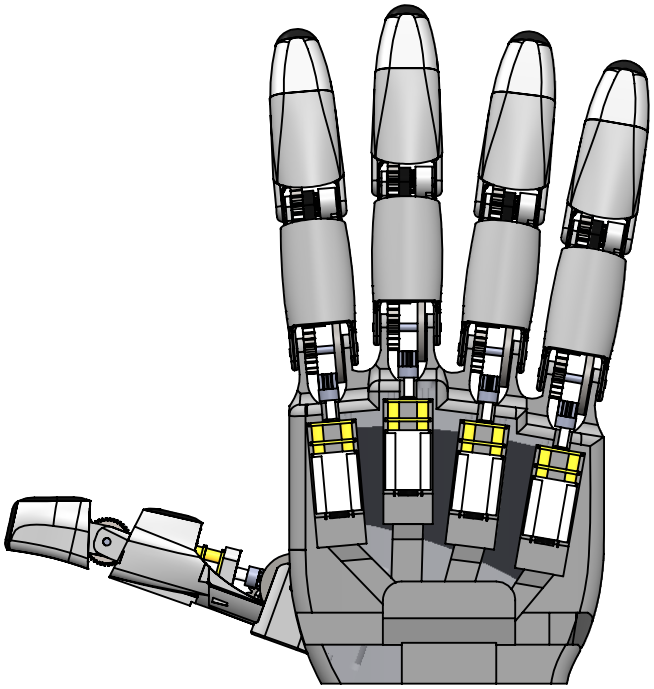


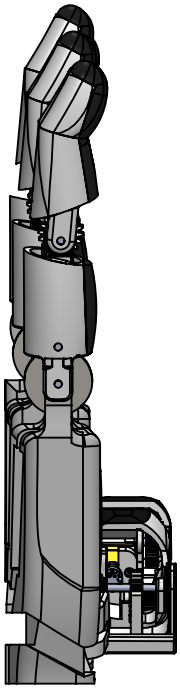


Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 1	Título: Ensamblaje de la mano robótica	
	Material: -	N.° de dibujo ENSM	Carta
	Unidades: mm	Escala: 1:2	Hoja 1 de 3

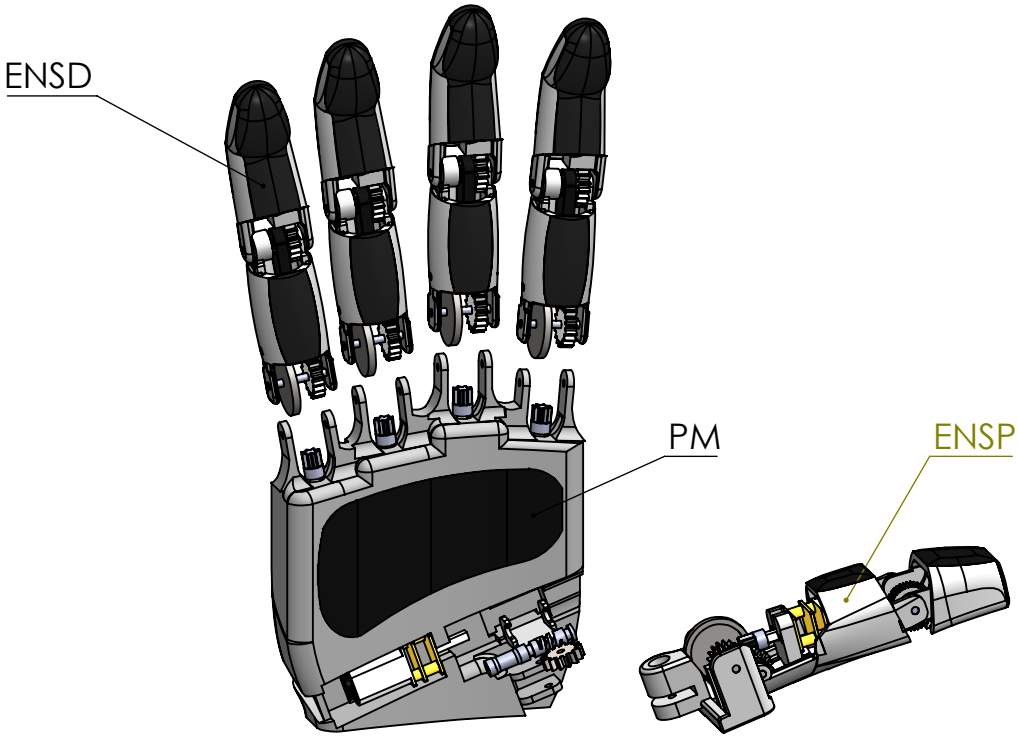






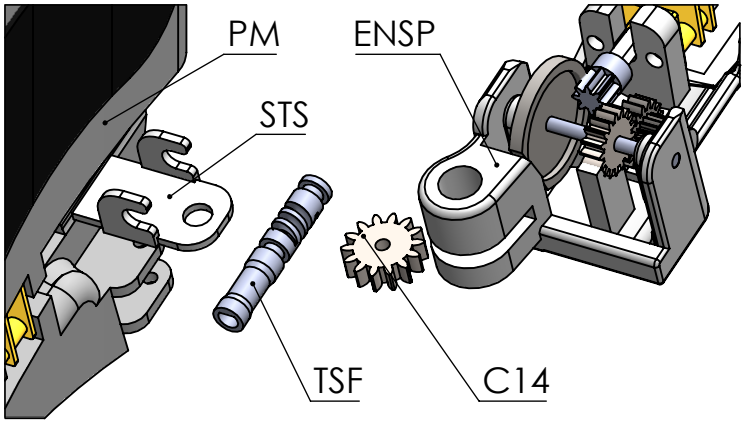


Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 1	Título: Ensambaje de la mano robótica	
	Material: -	N.° de dibujo ENSM	Carta
	Unidades: mm	Escala: 1:2	Hoja 2 de 3



Ensamble explosionado
del dedo pulgar con la palma

Escala 1:1



Elementos de la mano robótica

No. de dibujo	Título	Cant.
ENSD	Ensamblaje del dedo opuesto al pulgar	4
ENSP	Ensamblaje del dedo pulgar	1
TSF	Tornillo Sinfín	1
C14	Corona de 14 dientes	1
STS	Soporte para tortnillo sinfín	1
PM	Palma de la mano	1

Tesis: Desarrollo de un prototipo de prótesis activa de miembro superior utilizando señales mioeléctricas		Enero 2019 Rediseñado por: Leobardo Elí Sánchez Velasco	
Al menos que se indique lo contrario, las tolerancias para dimensiones de acabado deben ser de: 0.2 mm	Cantidad: 1 Material: - Unidades: mm	Título: Ensamblaje explosionado de la mano robótica N.° de dibujo EXPM Escala: 1:2	Carta Hoja 3 de 3

Apéndice B

Hoja de datos del microcontrolador ATmega328

8-bit AVR Microcontrollers

ATmega328/P

DATASHEET SUMMARY

Introduction

The Atmel® picoPower® ATmega328/P is a low-power CMOS 8-bit microcontroller based on the AVR® enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega328/P achieves throughputs close to 1MIPS per MHz. This empowers system designer to optimize the device for power consumption versus processing speed.

Feature

High Performance, Low Power Atmel®AVR® 8-Bit Microcontroller Family

- Advanced RISC Architecture
  - 131 Powerful Instructions
  - Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 20 MIPS Throughput at 20MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 32KBytes of In-System Self-Programmable Flash program Memory
  - 1KBytes EEPROM
  - 2KBytes Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data Retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
- Atmel® QTouch® Library Support
  - Capacitive Touch Buttons, Sliders and Wheels
  - QTouch and QMatrix® Acquisition
  - Up to 64 sense channels

- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Six PWM Channels
  - 8-channel 10-bit ADC in TQFP and QFN/MLF package
    - Temperature Measurement
  - 6-channel 10-bit ADC in PDIP Package
    - Temperature Measurement
  - Two Master/Slave SPI Serial Interface
  - One Programmable Serial USART
  - One Byte-oriented 2-wire Serial Interface (Philips I^2C compatible)
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - One On-chip Analog Comparator
  - Interrupt and Wake-up on Pin Change
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
- I/O and Packages
  - 23 Programmable I/O Lines
  - 28-pin PDIP, 32-lead TQFP, 28-pad QFN/MLF and 32-pad QFN/MLF
- Operating Voltage:
  - 1.8 - 5.5V
- Temperature Range:
  - -40°C to 105°C
- Speed Grade:
  - 0 - 4MHz @ 1.8 - 5.5V
  - 0 - 10MHz @ 2.7 - 5.5V
  - 0 - 20MHz @ 4.5 - 5.5V
- Power Consumption at 1MHz, 1.8V, 25°C
  - Active Mode: 0.2mA
  - Power-down Mode: 0.1µA
  - Power-save Mode: 0.75µA (Including 32kHz RTC)

1. Description

The Atmel AVR® core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in a single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega328/P provides the following features: 32Kbytes of In-System Programmable Flash with Read-While-Write capabilities, 1Kbytes EEPROM, 2Kbytes SRAM, 23 general purpose I/O lines, 32 general purpose working registers, Real Time Counter (RTC), three flexible Timer/Counters with compare modes and PWM, 1 serial programmable USARTs , 1 byte-oriented 2-wire Serial Interface (I2C), a 6-channel 10-bit ADC (8 channels in TQFP and QFN/MLF packages) , a programmable Watchdog Timer with internal Oscillator, an SPI serial port, and six software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or hardware reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption. In Extended Standby mode, both the main oscillator and the asynchronous timer continue to run.

Atmel offers the QTouch® library for embedding capacitive touch buttons, sliders and wheels functionality into AVR microcontrollers. The patented charge-transfer signal acquisition offers robust sensing and includes fully debounced reporting of touch keys and includes Adjacent Key Suppression® (AKS™) technology for unambiguous detection of key events. The easy-to-use QTouch Suite toolchain allows you to explore, develop and debug your own touch applications.

The device is manufactured using Atmel's high density non-volatile memory technology. The On-chip ISP Flash allows the program memory to be reprogrammed In-System through an SPI serial interface, by a conventional nonvolatile memory programmer, or by an On-chip Boot program running on the AVR core. The Boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega328/P is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications.

The ATmega328/P is supported with a full suite of program and system development tools including: C Compilers, Macro Assemblers, Program Debugger/Simulators, In-Circuit Emulators, and Evaluation kits.

2. Configuration Summary

Features	ATmega328/P
Pin Count	28/32
Flash (Bytes)	32K
SRAM (Bytes)	2K
EEPROM (Bytes)	1K
Interrupt Vector Size (instruction word/vector)	1/1/2
General Purpose I/O Lines	23
SPI	2
TWI (I^2C)	1
USART	1
ADC	10-bit 15kSPS
ADC Channels	8
8-bit Timer/Counters	2
16-bit Timer/Counters	1

and support a real Read-While-Write Self-Programming mechanism. There is a separate Boot Loader Section, and the SPM instruction can only execute from there. In , there is no Read-While-Write support and no separate Boot Loader Section. The SPM instruction can execute from the entire Flash.

3. Ordering Information

3.1. ATmega328

Speed [MHz][3]	Power Supply [V]	Ordering Code[2]	Package[1]	Operational Range
20	1.8 - 5.5	ATmega328-AU ATmega328-AUR[5] ATmega328-MMH[4] ATmega328-MMHR[4][5] ATmega328-MU ATmega328-MUR[5] ATmega328-PU	32A 32A 28M1 28M1 32M1-A 32M1-A 28P3	Industrial (-40°C to 85°C)

Note:

1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging, complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.
3. Please refer to *Speed Grades* for Speed vs. V_{CC}
4. Tape & Reel.
5. NiPdAu Lead Finish.

Package Type	
28M1	28-pad, 4 x 4 x 1.0 body, Lead Pitch 0.45mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
28P3	28-lead, 0.300” Wide, Plastic Dual Inline Package (PDIP)
32M1-A	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
32A	32-lead, Thin (1.0mm) Plastic Quad Flat Package (TQFP)

3.2. ATmega328P

Speed [MHz][3]	Power Supply [V]	Ordering Code[2]	Package[1]	Operational Range
20	1.8 - 5.5	ATmega328P-AU ATmega328P-AUR[5] ATmega328P-MMH[4] ATmega328P-MMHR[4][5] ATmega328P-MU ATmega328P-MUR[5] ATmega328P-PU	32A 32A 28M1 28M1 32M1-A 32M1-A 28P3	Industrial (-40°C to 85°C)
		ATmega328P-AN ATmega328P-ANR[5] ATmega328P-MN ATmega328P-MNR[5] ATmega328P-PN	32A 32A 32M1-A 32M1-A 28P3	Industrial (-40°C to 105°C)

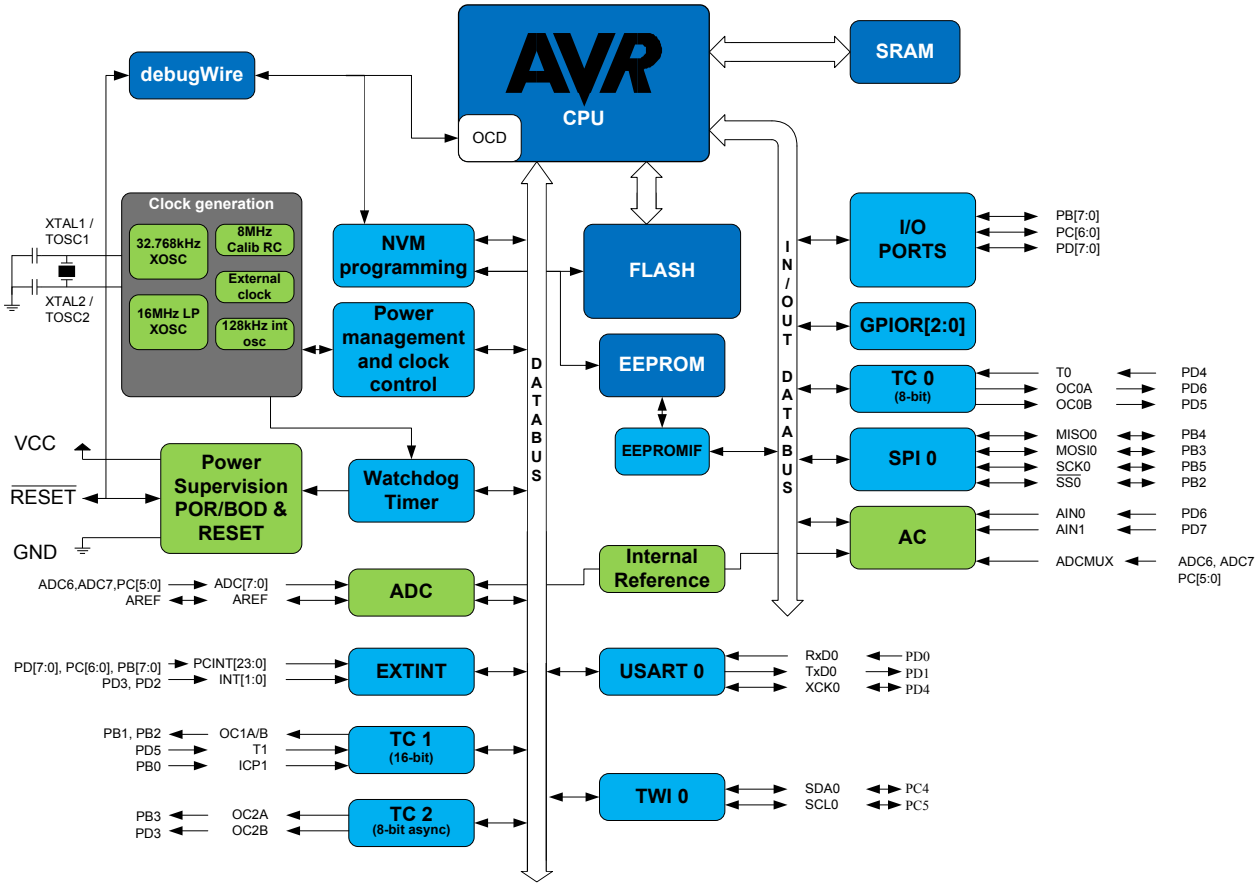
Note:

1. This device can also be supplied in wafer form. Please contact your local Atmel sales office for detailed ordering information and minimum quantities.
2. Pb-free packaging, complies to the European Directive for Restriction of Hazardous Substances (RoHS directive). Also Halide free and fully Green.
3. Please refer to *Speed Grades* for Speed vs. V_{CC}
4. Tape & Reel.
5. NiPdAu Lead Finish.

Package Type	
28M1	28-pad, 4 x 4 x 1.0 body, Lead Pitch 0.45mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
28P3	28-lead, 0.300” Wide, Plastic Dual Inline Package (PDIP)
32M1-A	32-pad, 5 x 5 x 1.0 body, Lead Pitch 0.50mm Quad Flat No-Lead/Micro Lead Frame Package (QFN/MLF)
32A	32-lead, Thin (1.0mm) Plastic Quad Flat Package (TQFP)

4. Block Diagram

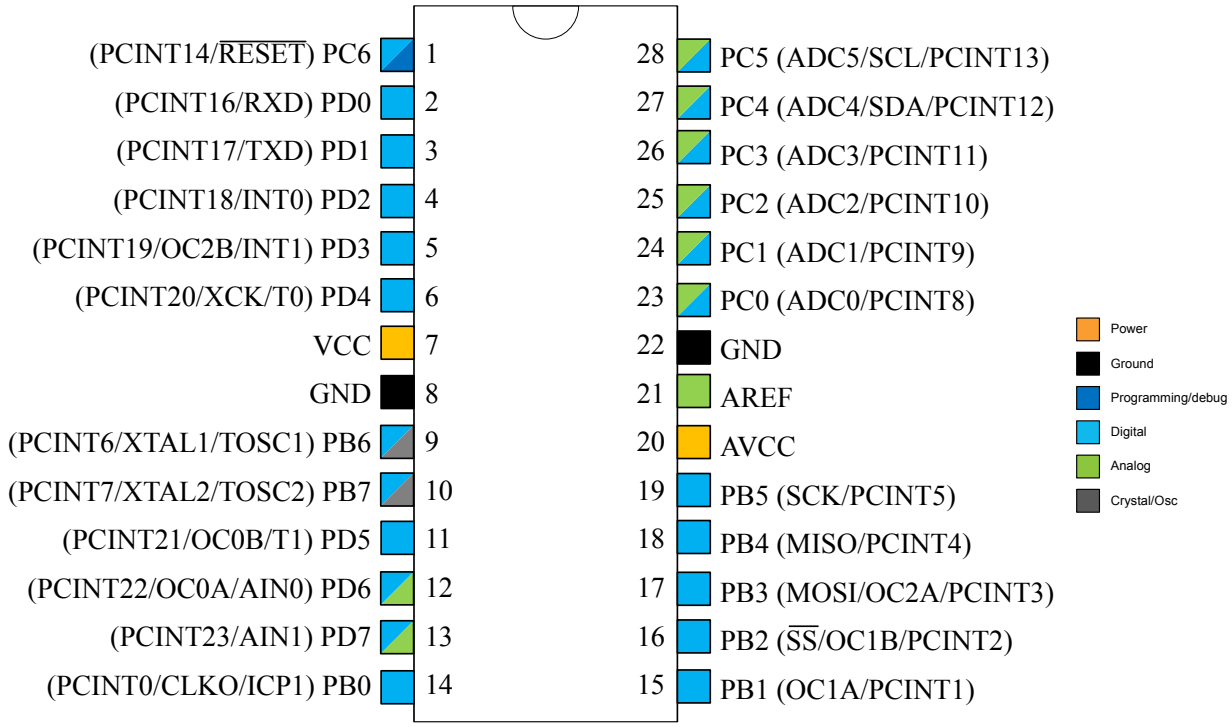
Figure 4-1. Block Diagram



5. Pin Configurations

5.1. Pin-out

Figure 5-1. 28-pin PDIP

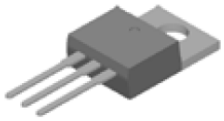


Apéndice C

Hoja de datos del regulador de voltaje LM7833

1A Positive Voltage Regulator

D-PACK (TO-252)



TO-220

General Description

- The TCI LM78XX family is monolithic fixed voltage regulator integrated circuit. They are suitable for applications that required supply current up to 1A.
- The LM78M is available in D-PACK (TO-252) and TO-220 packages.

.

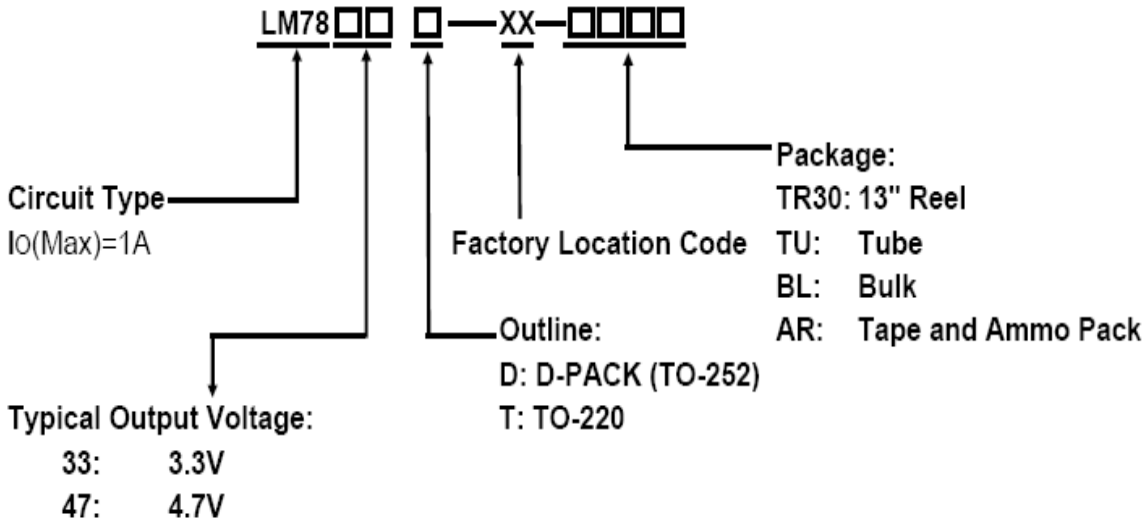
Features

- Output Current up to 1A
- Fixed output voltage of 3.3V and 4.7V available
- Thermal overload shutdown protection
- Short circuit current limiting
- Output transistor SOA protection
- RoHS Compliance

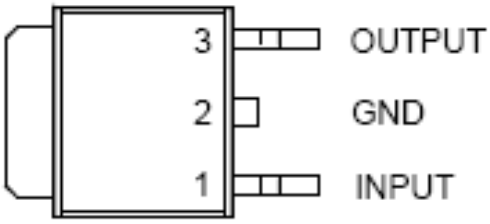
Applications

- High Efficiency Linear Regulator
- Post Regulation for Switching Supply
- Microprocessor Power Supply
- Mother Board

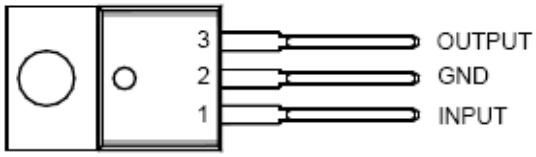
Ordering Information



Pin Configuration

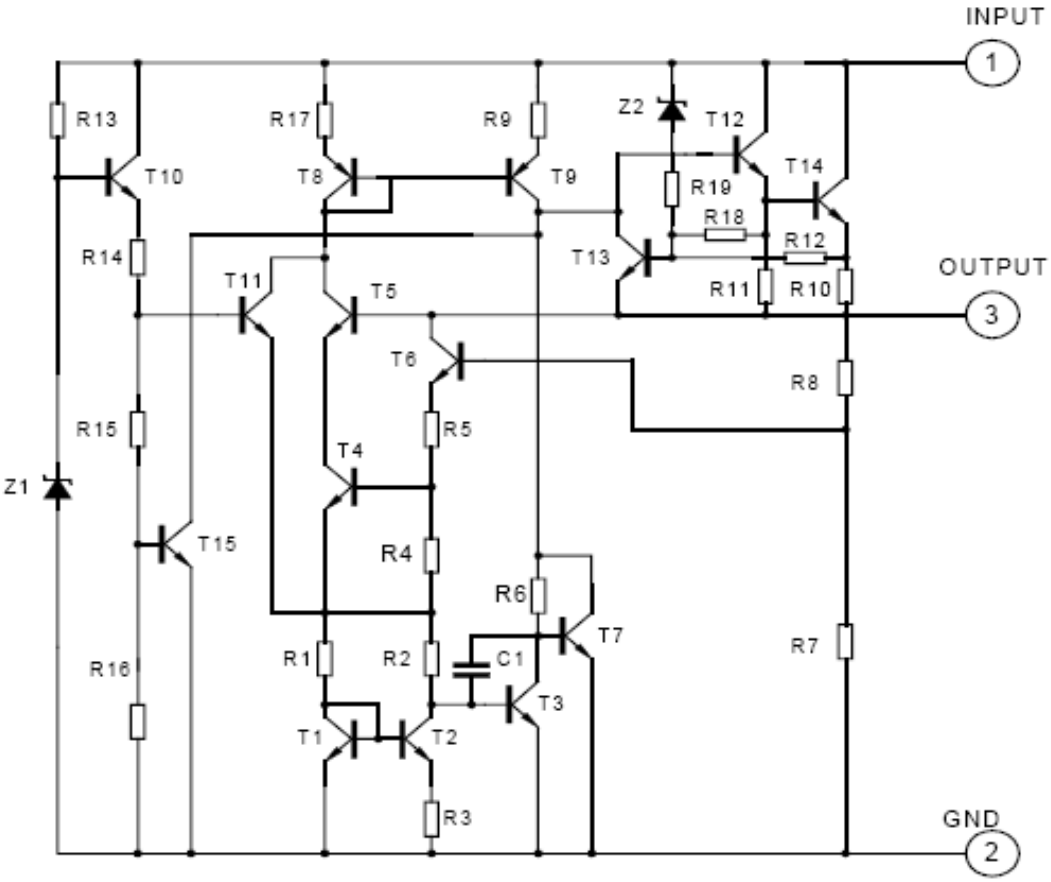


Outline: D
D-PACK
(TO-252)



Outline: T
TO-220

Block Diagram



Absolute Maximum Ratings

Symbol	Description		Ratings	Unit
V_{IN}	Input Voltage	V_{OUT}=3.3~18V	35	V
		V_{OUT}=20~24V	40	
I_{OUT}	Output Current		1	A
P_D	Power Dissipation	D-PACK (TO-252)	Internally Limited	mW
		TO-220		
T_J	Junction Temperature		150	
T_{OPR}	Operating Temperature Range		-20 ~ 150	° C
T_{STG}	Storage Temperature Range		-55 ~ 150	° C

Note: 1. Absolute maximum ratings are stress ratings only and functional device operation is not implied. The device could be damaged beyond Absolute maximum ratings.

2. The maximum steady state usable output current are dependent on input voltage, heat sinking, lead length of the package and copper pattern of PCB. The data are showed as electrical characteristics table represents pulse test conditions with junction temperatures specified at the initiation of test.

Electrical Characteristics (T_J=25° C, P_D≦15W, unless otherwise specified)

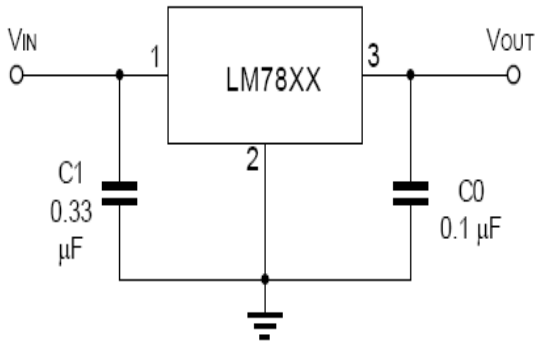
For LM7833 (V_{IN}=5.8V, I_{OUT}=0.5A, C1=0.33μF, C_O =0.1μF)

Symbol	Description	LM7833			Unit	Test Conditions
		Min.	Typ.	Max.		
V_{OUT}	Output Voltage	3.168	3.30	3.432	V	I_{OUT}=5mA-1.0A
		3.135	-	3.465	V	5.8V≤V_{IN}≤18.3V, I_{OUT}=5mA-1.0A
ΔV_{OUT}	Load Regulation	-	-	33	mV	I_{OUT}=5mA-1.0A
		-	-	17	mV	I_{OUT}=0.25A-0.75A
ΔV_{OUT}	Line Regulation	-	-	33	mV	5.8V≤V_{IN}≤18.3V
		-	-	33	mV	5.8V≤V_{IN}≤18.3V, I_{OUT}=1.0A
I_Q	Quiescent Current	-	-	8.0	mA	I_{OUT}≦1.0A
ΔI_Q	Quiescent Current Change	-	-	1.0	mA	5.8V≤V_{IN}≤18.3V
		-	-	0.5	mA	I_{OUT}=5mA-1.0A
e_N	Output Noise Voltage	-	55	-	μV	10Hz≤f≤100KHz
$\Delta V_O/\Delta T$	Temperature coefficient of V_{OUT}	-	-0.4	-	mV/℃	I_{OUT}=5mA
RR	Ripple Rejection	-	57	-	dB	6.3V≤V_{IN}≤16.3V, f=120Hz
I_{PEAK}	Peak Output Current	-	1.8	-	A	-
I_{SC}	Short-Circuit Current	-	250	-	mA	V_{IN}=35V
V_D	Dropout Voltage	-	2.0	-	V	-

For LM7847 (V_{IN}=9.7V, I_{OUT}=0.5A, C1=0.33μF, C_O =0.1μF)

Symbol	Description	LM7847			Unit	Test Conditions
		Min.	Typ.	Max.		
V_{OUT}	Output Voltage	4.512	4.70	4.888	V	I_{OUT}=5mA-1.0A
		4.465	-	4.935	V	7.2V≤V_{IN}≤19.7V, I_{OUT}=5mA-1.0A
ΔV_{OUT}	Load Regulation	-	-	47	mV	I_{OUT}=5mA-1.0A
		-	-	24	mV	I_{OUT}=0.25A-0.75A
ΔV_{OUT}	Line Regulation	-	-	47	mV	7.2V≤V_{IN}≤19.7V
		-	-	47	mV	7.2V≤V_{IN}≤19.7V, I_{OUT}=1.0A
I_Q	Quiescent Current	-	-	8.0	mA	I_{OUT}≦1.0A
ΔI_Q	Quiescent Current Change	-	-	1.0	mA	7.2V≤V_{IN}≤19.7V
		-	-	0.5	mA	I_{OUT}=5mA-1.0A
e_N	Output Noise Voltage	-	40	-	μV	10Hz≤f≤100KHz
$\Delta V_O/\Delta T$	Temperature coefficient of V_{OUT}	-	-0.6	-	mV/℃	I_{OUT}=5mA
RR	Ripple Rejection	62	80	-	dB	7.7V≤V_{IN}≤17.7V, f=120Hz
I_{PEAK}	Peak Output Current	-	1.8	-	A	-
I_{SC}	Short-Circuit Current	-	250	-	mA	V_{IN}=35V
V_D	Dropout Voltage	-	2.0	-	V	-

Typical Application



Note: Bypass capacitors are recommended for optimum stability and transient response and should be located as close as possible to the regulators.

Apéndice D

Código en Python ejecutado por la Raspberry Pi 3

```
'''
    Original by dzhu
        https://github.com/dzhu/myo-raw

    Edited by Fernando Cosentino
        http://www.fernandocosentino.net/pyoconnect
'''
from __future__ import print_function
import enum
import re
import struct
import sys
import threading
import RPi.GPIO as GPIO
import time
import serial
import extc
import os
import tty

from serial.tools.list_ports import comports
from common import *

contador=0
clasificador =False #True#False
captura = False
Diferentes = True
mayor = [0,0,0,0,0,0,0,0]
menor = [1000,1000,1000,1000,1000,1000,1000,1000]
promedio = [0, 0, 0, 0, 0, 0, 0, 0]
moda = [[[0],[0]], [[0],[0]], [[0],[0]], [[0],[0]],
```

```
32          [[0],[0]], [[0],[0]], [[0],[0]], [[0],[0]]]
33 maxi = [0,0,0,0,0,0,0,0]
34 veces = [0,0,0,0,0,0,0,0]
35 media= [0,0,0,0,0,0,0,0]
36 sensor = 0
37 no_datos = 100
38 Ndatosclasif = 40
39 media_aux = [[0],[0],[0],[0],[0],[0],[0],[0]]
40 modaclases = [0]
41
42 for y in range(49):
43     modaclases.append(0)
44 def nada():
45     print("-")
46
47 def puno():
48     GPIO.output(3,0)
49     GPIO.output(5,0)
50     GPIO.output(7,0)
51     GPIO.output(11,1)
52     print("Punio")
53
54 def descanso():
55     GPIO.output(3,1)
56     GPIO.output(5,0)
57     GPIO.output(7,0)
58     GPIO.output(11,1)
59     print("Descanso")
60
61 def cilindrico():
62     GPIO.output(3,0)
63     GPIO.output(5,1)
64     GPIO.output(7,0)
65     GPIO.output(11,1)
66     print("Cilindrico")
67
68 def punta():
69     GPIO.output(3,1)
70     GPIO.output(5,1)
71     GPIO.output(7,0)
72     GPIO.output(11,1)
73     print("Punta")
74
75 def gancho():
76     GPIO.output(3,0)
77     GPIO.output(5,0)
78     GPIO.output(7,1)
```

```
79      GPIO.output(11,1)
80      print("Gancho")
81
82  def palmar():
83      GPIO.output(3,1)
84      GPIO.output(5,0)
85      GPIO.output(7,1)
86      GPIO.output(11,1)
87      print("Palmar")
88
89  def esferico():
90      GPIO.output(3,0)
91      GPIO.output(5,1)
92      GPIO.output(7,1)
93      GPIO.output(11,1)
94      print("Esferico")
95
96  def lateral():
97      GPIO.output(3,1)
98      GPIO.output(5,1)
99      GPIO.output(7,1)
100     GPIO.output(11,1)
101     print("lateral")
102
103 GestoOutput = {0:nada, 1:descanso, 2:puno, 3:cilindrico,
104                4:punta, 5:gancho, 6:palmar, 7:esferico, 8:lateral}
105
106
107 for y in range(8):
108     for x in range(no_datos-1):
109         media_aux[y].append(0)
110
111 def multichr(ords):
112     if sys.version_info[0] >= 3:
113         return bytes(ords)
114     else:
115         return ''.join(map(chr, ords))
116
117 def multiord(b):
118     if sys.version_info[0] >= 3:
119         return list(b)
120     else:
121         return map(ord, b)
122
123 #EVENTOS DE TECLADO
124 from select import select
125
```

```
126 class NotTTYException(Exception): pass
127
128 class TerminalFile:
129     def __init__(self,infile):
130         if not infile.isatty():
131             raise NotTTYException()
132         self.file=infile
133
134         #prepare for getch
135         self.save_attr=tty.tcgetattr(self.file)
136         newattr=self.save_attr[:]
137         newattr[3] &= ~tty.ECHO & ~tty.ICANON
138         tty.tcsetattr(self.file, tty.TCSANOW, newattr)
139
140     def __del__(self):
141         #restoring stdin
142         import tty  #required this import here
143         tty.tcsetattr(self.file, tty.TCSADRAIN, self.save_attr)
144
145     def getch(self):
146         if select([self.file],[],[],0)[0]:
147             c=self.file.read(1)
148         else:
149             c=''
150         return c
151
152 class Arm(enum.Enum):
153     UNKNOWN = 0
154     RIGHT = 1
155     LEFT = 2
156
157 class XDirection(enum.Enum):
158     UNKNOWN = 0
159     X_TOWARD_WRIST = 1
160     X_TOWARD_ELBOW = 2
161
162 class Pose(enum.Enum):
163     REST = 0
164     FIST = 1
165     WAVE_IN = 2
166     WAVE_OUT = 3
167     FINGERS_SPREAD = 4
168     THUMB_TO_PINKY = 5
169     UNKNOWN = 255
170
171 class Packet(object):
172     def __init__(self, ords):
```

```
173         self.typ = ords[0]
174         self.cls = ords[2]
175         self.cmd = ords[3]
176         self.payload = multichr(ords[4:])
177 
178     def __repr__(self):
179         return 'Packet(%02X, %02X, %02X, [%s])' % \
180             (self.typ, self.cls, self.cmd,
181              ' '.join('%02X' % b for b in multiord(self.payload)))
182 
183 class BT(object):
184     '''Implements the non-Myo-specific details of the Bluetooth protocol.'''
185     def __init__(self, tty):
186         self.ser = serial.Serial(port=tty, baudrate=9600, dsrdtr=1)
187         self.buf = []
188         self.lock = threading.Lock()
189         self.handlers = []
190 
191     ## internal data-handling methods
192     def recv_packet(self, timeout=None):
193         t0 = time.time()
194         self.ser.timeout = None
195         while timeout is None or time.time() < t0 + timeout:
196             if timeout is not None:
197                 self.ser.timeout = t0 + timeout - time.time()
198             c = self.ser.read()
199             if not c:
200                 return None
201 
202             ret = self.proc_byte(ord(c))
203             if ret:
204                 if ret.typ == 0x80:
205                     self.handle_event(ret)
206                 return ret
207 
208     def recv_packets(self, timeout=.5):
209         res = []
210         t0 = time.time()
211         while time.time() < t0 + timeout:
212             p = self.recv_packet(t0 + timeout - time.time())
213             if not p: return res
214             res.append(p)
215         return res
216 
217     def proc_byte(self, c):
218         if not self.buf:
219             if c in [0x00, 0x80, 0x08, 0x88]:
```

```
220                 self.buf.append(c)
221             return None
222         elif len(self.buf) == 1:
223             self.buf.append(c)
224             self.packet_len = 4 + (self.buf[0] & 0x07) + self.buf[1]
225             return None
226         else:
227             self.buf.append(c)
228 
229         if self.packet_len and len(self.buf) == self.packet_len:
230             p = Packet(self.buf)
231             self.buf = []
232             return p
233         return None
234 
235     def handle_event(self, p):
236         for h in self.handlers:
237             h(p)
238 
239     def add_handler(self, h):
240         self.handlers.append(h)
241 
242     def remove_handler(self, h):
243         try: self.handlers.remove(h)
244         except ValueError: pass
245 
246     def wait_event(self, cls, cmd):
247         res = [None]
248         def h(p):
249             if p.cls == cls and p.cmd == cmd:
250                 res[0] = p
251         self.add_handler(h)
252         while res[0] is None:
253             self.recv_packet()
254         self.remove_handler(h)
255         return res[0]
256 
257     ## specific BLE commands
258     def connect(self, addr):
259         return self.send_command(6, 3, pack('6sBHHHH',
260                         multichr(addr), 0, 6, 6, 64, 0))
261 
262     def get_connections(self):
263         return self.send_command(0, 6)
264 
265     def discover(self):
266         return self.send_command(6, 2, b'\x01')
```

```
267
268     def end_scan(self):
269         return self.send_command(6, 4)
270
271     def disconnect(self, h):
272         return self.send_command(3, 0, pack('B', h))
273
274     def read_attr(self, con, attr):
275         self.send_command(4, 4, pack('BH', con, attr))
276         return self.wait_event(4, 5)
277
278     def write_attr(self, con, attr, val):
279         self.send_command(4, 5, pack('BHB', con, attr, len(val)) + val)
280         return self.wait_event(4, 1)
281
282     def send_command(self, cls, cmd, payload=b'', wait_resp=True):
283         s = pack('4B', 0, len(payload), cls, cmd) + payload
284         self.ser.write(s)
285
286         while True:
287             p = self.recv_packet()
288             ## no timeout, so p won't be None
289             if p.typ == 0: return p
290             ## not a response: must be an event
291             self.handle_event(p)
292
293
294 class MyoRaw(object):
295     '''Implements the Myo-specific communication protocol.'''
296
297     def __init__(self, tty=None):
298         if tty is None:
299             tty = self.detect_tty()
300         if tty is None:
301             raise ValueError('Myo dongle not found!')
302
303         self.bt = BT(tty)
304         self.conn = None
305         self.emg_handlers = []
306         self.imu_handlers = []
307         self.arm_handlers = []
308 #       self.pose_handlers = [] #leo
309         self.battery_handlers = []      #ALVIPE
310
311     def detect_tty(self):
312         for p in comports():
313             if re.search(r'PID=2458:0*1', p[2]):
```

```
314                 print('using device:', p[0])
315                 return p[0]
316
317         return None
318
319     def run(self, timeout=None):
320         self.bt.recv_packet(timeout)
321
322     def connect(self):
323         ## stop everything from before
324         self.bt.end_scan()
325         self.bt.disconnect(0)
326         self.bt.disconnect(1)
327         self.bt.disconnect(2)
328
329         ## start scanning
330         print('scanning...')
331         self.bt.discover()
332         while True:
333             p = self.bt.recv_packet()
334             print('scan response:', p)
335
336             if p.payload.endswith(b'\x06\x42\x48\x12\x4A\x7F\x2C\x48
337 ____________________________\x47\xB9\xDE\x04\xA9\x01\x00\x06\xD5'):
338                 addr = list(multiord(p.payload[2:8]))
339                 break
340         self.bt.end_scan()
341
342         ## connect and wait for status event
343         conn_pkt = self.bt.connect(addr)
344         self.conn = multiord(conn_pkt.payload)[-1]
345         self.bt.wait_event(3, 0)
346
347         ## get firmware version
348         fw = self.read_attr(0x17)
349         _, _, _, _, v0, v1, v2, v3 = unpack('BHBBHHHH', fw.payload)
350         print('firmware version: %d.%d.%d.%d' % (v0, v1, v2, v3))
351
352         self.old = (v0 == 0)
353
354         if self.old:
355             ## Myo Connect sends them, though we get data
356             ## fine without them
357             self.write_attr(0x19, b'\x01\x02\x00\x00')
358             self.write_attr(0x2f, b'\x01\x00')
359             self.write_attr(0x2c, b'\x01\x00')
360             self.write_attr(0x32, b'\x01\x00')
```

```
361             self.write_attr(0x35, b'\x01\x00')
362
363             ## enable EMG data
364             self.write_attr(0x28, b'\x01\x00')
365             ## enable IMU data
366             self.write_attr(0x1d, b'\x01\x00')
367             C = 1000
368             emg_hz = 50
369             ## strength of low-pass filtering of EMG data
370             emg_smooth = 100
371
372             imu_hz = 50
373
374             ## send sensor parameters, or we don't get any data
375             self.write_attr(0x19, pack('BBBBHBBBBB', 2, 9, 2, 1, C,
376                             emg_smooth, C // emg_hz, imu_hz, 0, 0))
377
378         else:
379             name = self.read_attr(0x03)
380             print('device name: %s' %name.payload)
381             ## enable IMU data
382             self.write_attr(0x1d, b'\x01\x00')
383             ## enable on/off arm notifications
384             self.write_attr(0x24, b'\x02\x00')
385             # self.write_attr(0x19, b'\x01\x03\x00\x01\x01')
386             self.start_raw()
387             #enable battery notifications
388             self.write_attr(0x12, b'\x01\x10')
389
390         ## add data handlers
391         def handle_data(p):
392             if (p.cls, p.cmd) != (4, 5): return
393
394             c, attr, typ = unpack('BHB', p.payload[:4])
395             pay = p.payload[5:]
396
397             if attr == 0x27:
398                 vals = unpack('8HB', pay)
399                 emg = vals[:8]
400                 moving = vals[8]
401                 self.on_emg(emg, moving)
402
403     # Read notification handles corresponding to the for EMG characteristics
404             elif attr == 0x2b or attr == 0x2e or attr == 0x31 or attr == 0x34:
405                 emg1 = struct.unpack('<8b', pay[:8])
406                 emg2 = struct.unpack('<8b', pay[8:])
407                 self.on_emg(emg1, 0)
```

```
408                 self.on_emg(emg2, 0)
409
410             # Read IMU characteristic handle
411             elif attr == 0x1c:
412                 vals = unpack('10h', pay)
413                 quat = vals[:4]
414                 acc = vals[4:7]
415                 gyro = vals[7:10]
416                 self.on_imu(quat, acc, gyro)
417             elif attr == 0x23:
418                 typ, val, xdir, _,_,_ = unpack('6B', pay)
419
420                 if typ == 1: # on arm
421                     self.on_arm(Arm(val), XDirection(xdir))
422                 elif typ == 2: # removed from arm
423                     self.on_arm(Arm.UNKNOWN, XDirection.UNKNOWN)
424                 #elif typ == 3: # pose          #leo
425                 #    self.on_pose(Pose(val)) #leo
426
427             # Read battery characteristic handle
428             elif attr == 0x11:
429                 battery_level = ord(pay)
430                 self.on_battery(battery_level)
431             else:
432                 print('data with unknown attr: %02X %s' % (attr, p))
433
434         self.bt.add_handler(handle_data)
435
436
437     def write_attr(self, attr, val):
438         if self.conn is not None:
439             self.bt.write_attr(self.conn, attr, val)
440
441     def read_attr(self, attr):
442         if self.conn is not None:
443             return self.bt.read_attr(self.conn, attr)
444         return None
445
446     def disconnect(self):
447         if self.conn is not None:
448             self.bt.disconnect(self.conn)
449
450     def sleep_mode(self, mode):
451         self.write_attr(0x19, pack('3B', 9, 1, mode))
452
453     def power_off(self):
454         self.write_attr(0x19, b'\x04\x00')
```

```
455
456     def start_raw(self):
457         self.write_attr(0x28, b'\x01\x00')
458         self.write_attr(0x19, b'\x01\x03\x01\x01\x01')
459
460     def mc_start_collection(self):
461         '''Myo Connect sends this sequence (or a reordering)
462         when starting data collection for v1.0 firmware;
463         this enables raw data but disables arm and pose
464         notifications.
465         '''
466         self.write_attr(0x28, b'\x01\x00')
467         self.write_attr(0x1d, b'\x01\x00')
468         self.write_attr(0x24, b'\x02\x00')
469         self.write_attr(0x19, b'\x01\x03\x01\x01\x01')
470         self.write_attr(0x28, b'\x01\x00')
471         self.write_attr(0x1d, b'\x01\x00')
472         self.write_attr(0x19, b'\x09\x01\x01\x00\x00')
473         self.write_attr(0x1d, b'\x01\x00')
474         self.write_attr(0x19, b'\x01\x03\x00\x01\x00')
475         self.write_attr(0x28, b'\x01\x00')
476         self.write_attr(0x1d, b'\x01\x00')
477         self.write_attr(0x19, b'\x01\x03\x01\x01\x00')
478
479     def mc_end_collection(self):
480         '''Myo Connect sends this sequence (or a reordering)
481         when ending data collection for v1.0 firmware;
482         this reenables arm and pose notifications, but
483         doesn't disable raw data.
484         '''
485         self.write_attr(0x28, b'\x01\x00')
486         self.write_attr(0x1d, b'\x01\x00')
487         self.write_attr(0x24, b'\x02\x00')
488         self.write_attr(0x19, b'\x01\x03\x01\x01\x01')
489         self.write_attr(0x19, b'\x09\x01\x00\x00\x00')
490         self.write_attr(0x1d, b'\x01\x00')
491         self.write_attr(0x24, b'\x02\x00')
492         self.write_attr(0x19, b'\x01\x03\x00\x01\x01')
493         self.write_attr(0x28, b'\x01\x00')
494         self.write_attr(0x1d, b'\x01\x00')
495         self.write_attr(0x24, b'\x02\x00')
496         self.write_attr(0x19, b'\x01\x03\x01\x01\x01')
497
498     def vibrate(self, length):
499         if length in xrange(1, 4):
500             ## first byte tells it to vibrate;
501             self.write_attr(0x19, pack('3B', 3, 1, length))
```

```
502     def set_leds(self, logo, line):
503         self.write_attr(0x19, pack('8B', 6, 6, *(logo + line)))
504 
505     def add_emg_handler(self, h):
506         self.emg_handlers.append(h)
507 
508     def add_imu_handler(self, h):
509         self.imu_handlers.append(h)
510 
511     def add_arm_handler(self, h):
512         self.arm_handlers.append(h)
513 
514     def add_battery_handler(self, h):
515         self.battery_handlers.append(h)
516 
517     def on_emg(self, emg, moving):
518         for h in self.emg_handlers:
519             h(emg, moving)
520 
521     def on_imu(self, quat, acc, gyro):
522         for h in self.imu_handlers:
523             h(quat, acc, gyro)
524 
525     def on_arm(self, arm, xdir):
526         for h in self.arm_handlers:
527             h(arm, xdir)
528 
529     def on_battery(self, battery_level):
530         for h in self.battery_handlers:
531             h(battery_level)
532 
533 if __name__ == '__main__':
534 
535     s=TerminalFile(sys.stdin)
536 
537 
538     try:
539         import pygame
540         HAVE_PYGAME = False
541     except ImportError:
542         HAVE_PYGAME = False
543 
544     if HAVE_PYGAME:
545         w, h = 800, 600 #w, h = 1200, 400
546         scr = pygame.display.set_mode((w, h))
547 
548     last_vals = None
```

```
549     def plot(scr, vals):
550         DRAW_LINES = True
551 
552         global last_vals
553         if last_vals is None:
554             last_vals = vals
555             return
556 
557         D = 5
558         scr.scroll(-D)
559         scr.fill((0,0,0), (w - D, 0, w, h))
560         for i, (u, v) in enumerate(zip(last_vals, vals)):
561             if DRAW_LINES:
562                 pygame.draw.line(scr, (0,255,0),
563                     (w - D, int(h/8 * (i+1 - u))),
564                     (w, int(h/8 * (i+1 - v))))
565                 pygame.draw.line(scr, (255,255,255),
566                     (w - D, int(h/8 * (i+1))),
567                     (w, int(h/8 * (i+1))))
568             else:
569                 c = int(255 * max(0, min(1, v)))
570                 scr.fill((c, c, c), (w - D, i * h / 8, D,
571                             (i + 1) * h / 8 - i * h / 8));
572 
573         pygame.display.flip()
574         last_vals = vals
575 
576     m = MyoRaw(sys.argv[1] if len(sys.argv) >= 2 else None)
577 
578     def proc_emg(emg, moving, times=[]):
579         if HAVE_PYGAME:
580             plot(scr, [e / 2000. for e in emg])
581 
582 
583         global clasificador, contador, captura, modaclases
584 
585         if clasificador :
586             #Exportacion de datos a la extension en C (extc)
587             #Clase determinada por algoritmo clasificador
588             #almacenada en la variable "clase"
589             clase = extc.clasi(emg[0],emg[1],emg[2],emg[3],
590                                emg[4],emg[5],emg[6],emg[7])
591             modaclases[contador]=clase
592             contador+=1
593 
594             if contador>= Ndatosclasif:
595                 #clasificador = False
```

```
596                 contador=0
597                 rep=0
598                 for x in modaclases:
599                     y = modaclases.count(x)
600                     if y>rep:
601                         rep=y
602                         modaGesto = x
603                 #print ("MODA:")
604                 GestoOutput[modaGesto]()
605                 #print (modaGesto)
606 
607         if captura & (contador <  no_datos):
608             global mayor, menor, promedio, moda, sensor, media_aux,media
609             print("Capturando dato numero", contador+1)
610             print (emg)
611             for sensor in range(8):
612                 media_aux [sensor] [contador] = emg[sensor]
613                 if mayor[sensor] < emg[sensor]:
614                     mayor[sensor] = emg[sensor]
615                 if menor[sensor] > emg[sensor]:
616                     menor[sensor] = emg[sensor]
617 
618                 promedio[sensor] += emg[sensor]
619 
620                 if (emg[sensor] in moda[sensor][0]):
621                     pos = moda[sensor][0].index(emg[sensor])
622                     while (len(moda[sensor][1]) <= pos):
623                         moda[sensor][1].append(0)
624                     moda[sensor][1][pos] += 1
625                 else:
626                     moda[sensor][0].append(emg[sensor])
627                     moda[sensor][1].append(1)
628 
629             contador +=1
630             if contador >= no_datos:
631                 m.vibrate(1)
632                 for x in range(len (promedio)):
633                     promedio[x] = promedio[x]/no_datos
634                     for x in range(8):
635                         media_aux[x].sort()
636                         media[x] = (media_aux[x][no_datos/2]
637                                 + media_aux[x][(no_datos/2) + 1])/2
638 
639 
640                 print("-----------------------------------------------")
641                 print(" La captura de datos ha finalizado" )
642                 print("Los datos capturados son:")
```

```
643                 for x in range(8):
644                     print("Valor  #Veces ------Senesor:", x+1)
645                     for y in range(len(moda[x][0])):
646                         if moda[x][1][y] > 0:
647                             print(moda[x][0][y] ,"-----", moda[x][1][y])
648                         if moda[x][1][y] > veces[x]:
649                             veces[x]=moda[x][1][y]
650                             maxi[x]=moda[x][0][y]
651 
652                 print("-------------------------------------------------")
653                 print(" Numero de datos capturados:")
654                 print(contador)
655                 print("Promedio:")
656                 print(promedio)
657                 print("Media:")
658                 print(media)
659                 print(" Mayor:")
660                 print(mayor)
661                 print(" Menor:")
662                 print(menor)
663                 print(" Moda:")
664                 print(maxi)
665                 print(veces)
666                 contador = 0
667                 captura = False
668 
669         # print framerate of received data
670         times.append(time.time())
671         if len(times) > 20:
672             #print((len(times) - 1) / (times[-1] - times[0]))
673             times.pop(0)
674 
675     def proc_battery(battery_level):
676         print("Battery level: %d" % battery_level)
677         if battery_level < 5:
678             m.set_leds([255, 0, 0], [255, 0, 0])
679         else:
680             m.set_leds([128, 128, 255], [128, 128, 255])
681 
682 
683     m.add_emg_handler(proc_emg)
684     m.add_battery_handler(proc_battery)
685     m.connect()
686 
687     m.add_arm_handler(lambda arm, xdir: print('arm', arm, 'xdir', xdir))
688 
689     m.sleep_mode(1)
```

```
690     m.vibrate(1)
691
692
693 #Configuración puerto GPIO de la Raspberry
694     GPIO.setmode(GPIO.BOARD)
695     GPIO.setup(3,GPIO.OUT)
696     GPIO.setup(5,GPIO.OUT)
697     GPIO.setup(7,GPIO.OUT)
698     GPIO.setup(11,GPIO.OUT)
699
700
701     try:
702         print ("Presiona q para salir...")
703         print ("Presiona s para calcular pose")
704         print ("Presiona c para iniciar el clasificador")
705         GPIO.output(11,0)
706         while True:
707             m.run(1)
708             teclado = s.getch()
709             if (teclado == " "):
710                 continue
711             elif (teclado == "q"):
712                 raise KeyboardInterrupt()
713             elif (teclado == "x"):
714                 captura = False
715                 clasificador = False
716                 contador=0
717                 print ("Presiona q para salir...")
718                 print ("Presiona s para calcular pose")
719                 print ("Presiona c para iniciar el clasificador")
720             elif (teclado == "s"):
721                 print("Iniciando captura de datos: ")
722                 captura = True
723                 clasificador = False
724                 contador=0
725
726             elif (teclado == "c"):
727                 print("Iniciando clasificador... ")
728                 clasificador = True
729                 contador=0
730
731
732             if HAVE_PYGAME:
733                 for ev in pygame.event.get():
734                     if ev.type == QUIT or
735                         (ev.type == KEYDOWN and ev.unicode == 'q'):
736                         raise KeyboardInterrupt()
```

```
737                     elif (ev.type == KEYDOWN and ev.unicode == 's'):
738                         print("Iniciando captura de datos: ")
739                         captura = True
740 
741                     elif (ev.type == KEYDOWN and ev.unicode == 'c'):
742                         print("Iniciando clasificador ... ")
743                         clasificador = True
744 
745                     elif ev.type == KEYDOWN:
746                         if K_1 <= ev.key <= K_3:
747                             m.vibrate(ev.key - K_0)
748                         if K_KP1 <= ev.key <= K_KP3:
749                             m.vibrate(ev.key - K_KP0)
750 
751 
752     except KeyboardInterrupt:
753         pass
754     finally:
755 
756         GPIO.output(11,0)
757         m.vibrate(1)
758         m.disconnect()
759         print("Desconectado")
760         GPIO.cleanup()
```

Apéndice E

Código de la extensión Python-C encargada de la clasificación

```
#include <Python.h>
#include <stdio.h>
#include <math.h>
#define NumClases 8
#define NumSensores 8

static PyObject *MAE(PyObject *self, PyObject *args);

static PyMethodDef ClasiMethods[] = //Declarar métodos
{
    //"PythonName"  C-function Name, argument presentation, description
    {"clasi",MAE,METH_VARARGS,  "Memoria Asociativa Extendida"},
    {NULL, NULL, 0, NULL}        /* Sentinel */
};

void initextc(void)
{
(void) Py_InitModule("extc", ClasiMethods);
}

static PyObject *MAE(PyObject *self, PyObject *args)
{
int emg[NumSensores];
int error[NumClases][NumSensores];
int mayor = 0;
int mayores [NumClases];
int menor = 1000;
int clase;
int i,j;
int clases[NumClases][NumSensores]= {
    {27, 34, 59, 51, 64, 34, 27, 23},        // descanso
```

```
32      {353, 211, 605, 783, 123, 434, 420, 785},// punio
33      {354, 201, 89, 128, 179, 192, 637, 394}, // cilindrico (palma adentro)
34      {79, 106, 234, 897, 704, 262, 123, 142}, // punta (palma afuera)
35      {297, 99, 105, 526, 440, 217, 150, 618}, // gancho (spock)
36      {150, 247, 700, 414, 74, 164, 420, 316}, // palmar (pist2)
37      {65, 68, 144, 348, 253, 59, 43, 51},     // esferico (indice-pulg)
38      {90, 106, 318, 523, 458, 456, 139, 158}, // lateral (ok afuera)
39  };
40
41  //Convertir de Python a C
42  if (!PyArg_ParseTuple(args, "iiiiiiii",&emg[0],&emg[1],&emg[2],&emg[3],
43                                         &emg[4],&emg[5],&emg[6],&emg[7]))
44      {return NULL;}
45
46  /// min ( max ( clases[][] - emg[]))
47
48      for (i = 0; i< NumClases; i++)
49      {
50          for (j = 0; j< NumSensores; j++)
51          {
52              error[i][j]=fabs (clases[i][j] - emg[j]);
53
54              if (error [i][j] > mayor)
55              {
56                  mayor = error[i][j];
57                  mayores [i] = mayor;
58              }
59          }
60          mayor = 0;
61          if (mayores[i] < menor)
62          {
63              clase = i+1;
64              menor = mayores[i];
65          }
66      }
67
68  if (menor > 300)     //Nada
69  {
70      clase=0;
71  }
72  return Py_BuildValue("i",clase ); //Convertir y retornar dato a Python
73  }
```

Apéndice F

Código ejecutado por el microcontrolador

```
#define F_CPU 8000000UL
#include "delay8MHZ.h"
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdlib.h>
#define TiempoMuerto 2000

volatile uint8_t agarre = 0;
void ActivarPWM( uint8_t motor,uint8_t PWM, uint8_t enciendido);
void AjustarDireccion(uint8_t direccion, uint8_t motor);
void configPWM(void);

int main(void){

    uint8_t encoderPrev[6];
    uint8_t encoderAct[6];
    uint8_t encoder1;
    uint8_t encoder2;
    uint8_t n;
    uint8_t agarrePrev=0;
    uint8_t motor = 6;
    uint8_t direccion[6] = {0,0,0,0,0,0};
    uint8_t activar=1;
    uint8_t PWM=0;
    uint8_t espera=0;
    uint8_t CambioMotor=1;
    uint8_t CambioGesto=0;
    uint8_t RasPi;


    int pos[6] = {0,0,0,0,0,273};//Numero de Flancos
    float posicion[6] = {0,0,0,0,0,0};
    float error=0;
```

```
34
35      int ciclos=0;
36      int gesto[6] = {0,0,0,0,0,0};
37  //  DEDO     { Menique,  Anular, Medio, Indice, Palma, Pulgar}
38  //  MOTOR:       { 0       1       2       3       4       5}
39  int gesto0[6] = {  0,     0,      0,      0,      0,     23};//punio
40  int gesto1[6] = {100,   100,    100,    100,    110,     94};//Descanso
41  int gesto2[6] = { 58,    53,     50,     55,     15,     69};//Cilindrico
42  int gesto3[6] = {  0,     0,      0,     47,     50,     33};//Punta
43  int gesto4[6] = { 30,    28,     25,     25,    110,    94}; //Gancho
44  int gesto5[6] = {  0,     0,     36,     45,     40,     39};//Palmar
45  int gesto6[6] = { 32,    44,     40,     31,     15,     60};//Esferico
46  int gesto7[6] = {  0,    10,     20,     30,    100,     26};//lateral
47  int gesto8[6] = {  0,    10,     20,     30,    105,     28};//Prueba
48
49
50      //Entradas - Saliads
51      DDRB = 0x0E; // 0000 1110 PWM Y ENCODERS (libres: -)
52      DDRC = 0x3F; // 0011 1111 ADC DIRECCIoN DE MOTORES (libres: -PC6)
53      DDRD = 0x68; // 0110 1000 PWM PUSH
54      PORTD = 0X04; //  0000 0100 Pull up en PD2 (PUSH)
55
56      configPWM();      //configura PWM
57
58      sei();            //Habilitador global de interrupciones.
59
60      //INICIALIZAR ENCODES
61      encoder1 = PINB;      //Lee PUERTO B (PB4 PB5 PB6 PB7 PB0)
62      encoder2 = PIND;      //Lee PUERTO D (PD7)
63      encoderPrev[0] = (encoder1 & (1<<PB4)) ? 1 : 0;
64      encoderPrev[1] = (encoder1 & (1<<PB5)) ? 1 : 0;
65      encoderPrev[2] = (encoder1 & (1<<PB6)) ? 1 : 0;
66      encoderPrev[3] = (encoder1 & (1<<PB7)) ? 1 : 0;
67      encoderPrev[4] = (encoder1 & (1<<PB0)) ? 1 : 0;
68      encoderPrev[5] = (encoder2 & (1<<PD7)) ? 1 : 0;
69
70
71      while(1)
72      {
73          RasPi=PIND;                     //leer PD0-PD4 para leer agarre
74          if (RasPi & (1<<PD4))
75          {
76              (RasPi & (1<<PD0))? (agarre |= (1<<0)) : (agarre &=~ (1<<0));
77              (RasPi & (1<<PD1))? (agarre |= (1<<1)) : (agarre &=~ (1<<1));
78              (RasPi & (1<<PD2))? (agarre |= (1<<2)) : (agarre &=~ (1<<2));
79          }
80          else
```

```
81              agarre = 0; // Punio
82
83          {   //Leer PINES para establecer estado de encoderAct[0-5]
84              encoder1 = PINB;
85              encoder2 = PIND;
86              encoderAct[0] = (encoder1 & (1<<PB4)) ? 1 : 0;
87              encoderAct[1] = (encoder1 & (1<<PB5)) ? 1 : 0;
88              encoderAct[2] = (encoder1 & (1<<PB6)) ? 1 : 0;
89              encoderAct[3] = (encoder1 & (1<<PB7)) ? 1 : 0;
90              encoderAct[4] = (encoder1 & (1<<PB0)) ? 1 : 0;
91              encoderAct[5] = (encoder2 & (1<<PD7)) ? 1 : 0;
92          }
93
94          if (motor != 6) //Leer posiciones de encoders
95          {
96              //Deteccion de flancos
97              if (((encoderPrev[motor] == 0) && (encoderAct[motor] == 1))
98                  || ((encoderPrev[motor] == 1) && (encoderAct[motor] == 0)))
99              {
100                 if (direccion[motor]==1) // si el dedo se abre
101                     pos[motor]++;        //aumenta la posicion de encoder
102                 else                  //si el dedo se cierra
103                     pos[motor]--;        //disminuye la posicion de encoder
104
105
106                 if (motor==4) //Para la palma
107                 {//recalcula posicion en pasos PARA LA PALMA
108                     posicion[motor] = ((pos[motor]*0.4) / 14) ;
109                 } else //Para el resto de los dedos
110                 {//recalcula posicion en grados
111                     posicion[motor] = (pos[motor] * 0.4)/ (4.5);
112                 }
113             }encoderPrev[motor] = encoderAct[motor];
114         }
115
116         //detectar cambio de pose
117         if ((agarrePrev != agarre) && (CambioMotor==1))
118         {   //Cambia de agarre?
119             motor = 5;                    // MOTOR INICIAL
120             agarrePrev = agarre;
121             CambioGesto=1;                // mover primero el motor 5
122             CambioMotor=0;
123
124             switch (agarre){     // Seleccion de posiciones para cada gesto
125                 case 0:
126                     for(n=0;n<6;n++)
127                     {    gesto[n]=gesto0[n];}
```

```
128                     break;
129                 case 1:
130                     for(n=0;n<6;n++)
131                     {   gesto[n]=gesto1[n];}
132                     break;
133                 case 2:
134                     for(n=0;n<6;n++)
135                     {   gesto[n]=gesto2[n];}
136                     break;
137                 case 3:
138                     for(n=0;n<6;n++)
139                     {   gesto[n]=gesto3[n];}
140                     break;
141                 case 4:
142                     for(n=0;n<6;n++)
143                     {   gesto[n]=gesto4[n];}
144                     break;
145                 case 5:
146                     for(n=0;n<6;n++)
147                     {   gesto[n]=gesto5[n];}
148                     break;
149                 case 6:
150                     for(n=0;n<6;n++)
151                     {   gesto[n]=gesto6[n];}
152                     break;
153                 case 7:
154                     for(n=0;n<6;n++)
155                     {   gesto[n]=gesto7[n];}
156                     break;
157                 case 8:
158                     for(n=0;n<6;n++)
159                     {   gesto[n]=gesto8[n];}
160                     break;
161             }
162         }
163 
164         //Controlar posicion
165         if (motor != 6)
166         {
167             CambioMotor=0;
168 
169             if ((motor==5)&&(CambioGesto==1))
170             {
171                 //abrir pulgar al iniciar
172                 error = posicion[motor] - 95;
173             }
174             else
```

```
175                 //Calcula el error con el gesto deseado (agarre)
176                 {error = posicion[motor] - gesto[motor];}
177
178             //Controlar PWM dependiendo del valor del error
179             if(fabsf(error) > 15 )
180             {   PWM=255;}
181                 else if (fabsf(error) >5 )
182                     {   PWM=150;    }
183                     else
184                     {   PWM=75; }
185
186             if (motor==4)
187                 PWM=250;
188
189             if (fabsf(error) <= 1)
190             {//si el error es cero Se apaga todo y se activa modo espera
191                 PWM=0;
192                 activar=0;
193                 espera=1;
194             }
195
196             if (espera==0)      //Modo espera desactivado
197             {//direccion de acuerdo a la direccion del driver y activar
198                 if (error < 0)
199                     {   direccion[motor] = 1;}
200                 else
201                     {   direccion[motor] = 0; }
202
203                 activar=1;
204             }
205             else                //Modo espera activado
206             {
207                 ciclos++;
208                 if (ciclos > TiempoMuerto)
209                 {
210                     ciclos=0;
211                     espera=0;
212                     CambioMotor=1;
213
214                     switch (motor){
215                         case 0:
216                             motor=1;
217                             break;
218                         case 1:
219                             motor=2;
220                             break;
221                         case 2:
```

```
222                                 motor=3;
223                                 break;
224                             case 3:
225                                 motor=5;
226                                 break;
227                             case 4:
228                                 motor=0;
229                                 break;
230                             case 5:
231                                 if (CambioGesto==1)
232                                 {   CambioGesto=0;
233                                     motor=4;}
234                                 else
235                                 {motor=6;}
236                                 break;
237 
238                             default: break;
239 
240                         }
241                     }
242                 }
243 
244                 if (motor!=6)
245                 {   AjustarDireccion(direccion[motor], motor);
246                     ActivarPWM(motor, PWM, activar);
247                 }
248             }
249         }
250 }
251 //Motor a activar // PWM //encendido 1=ENCENDER PWM, 0=APAGAR PWM
252 void ActivarPWM( uint8_t motor,uint8_t PWM, uint8_t enciendido)
253 {
254     switch (motor){
255         case 0:
256         OCR0A=PWM;
257         if (enciendido==1)
258         {TCCR0A |= (1<<7);}
259         else
260         {TCCR0A &=~ (1<<7);}
261         break;
262 
263         case 1:
264         OCR0B=PWM;
265         if (enciendido==1)
266         {TCCR0A |= (1<<5);}
267         else
268         {TCCR0A &=~ (1<<5);}
```

```
269             break;
270
271             case 2:
272             OCR1A=PWM;
273             if (enciendido==1)
274             {TCCR1A |= (1<<7);}
275             else
276             {TCCR1A &=~ (1<<7);}
277             break;
278
279             case 3:
280             OCR1B=PWM;
281             if (enciendido==1)
282             {TCCR1A |= (1<<5);}
283             else
284             {TCCR1A &=~ (1<<5);}
285             break;
286
287             case 4:
288             OCR2A=PWM;
289             if (enciendido==1)
290             {TCCR2A |= (1<<7);}
291             else
292             {TCCR2A &=~ (1<<7);}
293             break;
294
295             case 5:
296             OCR2B=PWM;
297             if (enciendido==1)
298             {TCCR2A |= (1<<5);}
299             else
300             {TCCR2A &=~ (1<<5);}
301             break;
302
303             default:
304             break;
305         }
306 }
307
308 //Ajuste de direccion de motores
309 void AjustarDireccion(uint8_t direccion, uint8_t motor)
310 {
311 if (motor < 4)
312     (direccion==1)? (PORTC |= (1<<motor)) : (PORTC &=~ (1<<motor));
313 else
314     (direccion==1)? (PORTC &=~ (1<<motor)) : (PORTC |= (1<<motor));
315 }
```

```
316
317 //Configuracion de PWM
318 void configPWM(void)
319 {
320     //Config PWM0
321     TCCR0A=0x03;     // 1010 0011 PWM rapido no invertido
322     TCCR0B=0x02;     // 0000 0010 escalador /8
323
324     //Config PWM1
325     TCCR1A=0x01;     //A1 1010 0001 PWM rapido no invertido
326     TCCR1B=0X0A;     // 0000 1010 normal, escalador /8
327
328     //Config PWM2
329     TCCR2A=0x03;     //A3 1010 0011 PWM rapido no invertido
330     TCCR2B=0X02;     // 0000 0010 normal, escalador /8
331     //-------------------------------------------------------
332 }
```

Bibliografía

[1] Vázquez, V. E., Hiiar, M. M. *Los amputados, un reto para el estado*. Acta de la Sesión del 4 de marzo del 2015 Academia Nacional de Medicina, México.

[2] Weir, R. F. *Design of artificial arms and hands for prosthetic applications*. Standard Handbook of Biomedical Engineering and Design. M.Kutz, Ed., New York, NY: McGraw Hill, 2003, pp.32.1 - 32.61.

[3] Universidad de Valencia *Tema 6. Prótesis*. Protésica. Obtenido de: https://www.uv.es/mpisea/ndice_de_temas.html, Consultado en Enero de 2018.

[4] Díaz, L. I *Diseño y construcción de un socket de miembro superior con suspensión ajustable*. Tesis de Maestría en Mecánica, Universidad Nacional Auntónoma de México, México, Enero 2010.

[5] Protésica *Prótesis de miembro superior*. Obtenido de: http://protesica.com.co/protesis-de-miembro-superior/ Consultado en Enero de 2018.

[6] Dorador, M. *Robótica y prótesis inteligentes*. Revista Digital Universitaria, Vol. 6, Núm. 1; Universidad Nacional Auntónoma de México, México, Enero 2004.

[7] Brito, J., Quinde, X., Cusco, D., Calle, J. I. *Estudio del estado del arte de las prótesis de mano*. INGENIUS Núm. 9, (Enero-Junio). Universidad Politécnica Salesiana, Ecuador, 2013.

[8] Ortosur *Catálogo de manos eléctricas*. Obtenido de: http://www.ortosur.net/melectricas.html Consultado en Enero de 2018.

[9] Universidad de Valencia *Tema 1. Introducción a la ortoprotésica* . Protésica. Obtenido de: https://www.uv.es/mpisea/ndice_de_temas.html, Consultado en Enero de 2018.

[10] García, V., García, M., Hernández, K. *Diseño de prótesis mioeléctrica*. Diseño mecánico y de control, Tesis de maestría, Instituto Politécnico Nacional, México, 2004.

[11] Shadow Robot Company *Shadow Dexterous Hand*. Obtenido de: https://www.shadowrobot.com/products/dexterous-hand/ Consultado en Enero de 2018.

[12] CIDOP Ortopedia *Prótesis - Miembro Superior*. Obtenido de: https://www.cidoportopedia.com/protesisms Consultado en Enero de 2018.

[13] Quinayás, B. C *Diseño y construcción de una prótesis robótica de mano funcional adaptada a varios agarres*. Tesis de Maestría en Automática, Universidad de Cauca, Popayán, Colombia, Enero 2010.

[14] Norton, K. *A Brief History of Prosthetics*. Revista inMotion, Vol. 12, Núm. 7, pp 11-13; Amputee Coalition, Estados Unidos, Diciembre 2007.

[15] Ottobock *Sistema de mano protésica MyoFacil*. Obtenido de: http://www.ottobock.es/protesica/miembro-superior/sistemas-de-brazo-y-mano/sistema-protesico-myofacil/ Consultado en Enero de 2018.

[16] Ottobock *Prótesis de mano Michelangelo*. Obtenido de: http://www.ottobock.es/protesica/miembro-superior/sistemas-de-brazo-y-mano/axon-bus-con-mano-michelangelo/ Consultado en Enero de 2018.

[17] Touchbionics *i-limb ultra*. Obtenido de: https://www.touchbionics.com/products/active-prostheses/i-limb-ultra Consultado en Enero de 2018.

[18] Bebionic *Características del bebionic 3*. Obtenido de: http://es.bebionic.com/the_hand/features/ Consultado en Enero de 2018.

[19] Probionics *Diseño y desarrollo de órganos artificiales*. Obtenido de: http://www.probionics.com.mx Consultado en Febrero de 2018.

[20] Krausz, N., Rorrer, R., Weir, R. *Design and Fabrication of a Six Degree-of-Freedom Open Source Hand*. IEEE Transactions on Neural Systems and Rehabilitation Engineering, Vol. 24, Núm. 5, pp. 562 - 572. IEEE Engineering in Medicine and Biology Society, Mayo 2016.

[21] Norton, L. *Diseño de máquinas. Un enfoque integrado*. Cuarta edición,Ed. Pearson Educación, México, 2011, pp. 558-560.

[22] Budynas, R., Nisbett, J. *Diseño en ingeniería mecánica de Shigley*. Octava edición, Ed. Mc Graw Hill, México, 2008, pp. 654-680, 860-887.

[23] Taylor, C., Schwarz, R. *The Anatomy and Mechanics of the Human Hand*. Artificial Limbs: A Review of Current Developments, Vol. 2, Núm. 2, 1955, pp. 22-35.

[24] Gómez, J. *La electromiografía: Un acercamiento al concepto fisiológico, la construcción de un equipo electromiográfico con registro no invasivo; y la resistencia galvánica de piel como método de relajación muscular.*. Universidad Tecnológica de Pereira, Facultad de Ciencias Básicas, Maestría en instrumentación física, Colombia, 2009.

[25] Massó, N., Rey, F., Romero, D., Gual, G., Costa, L., Germán, A. *Surface electromyography applications in the sport*. Apunts Med Esport, Vol. 45, Núm. 165, pp. 121 - 130. Universidad Ramon Llull, Facultad de Ciencias de la Salud Blanquerna, Barcelona, España, 2010.

[26] Sucar, L. *Clasificadores Bayesianos: de Datos a Conceptos*. Instituto Nacional de Astrofísica, Óptica y Electrónica, Tonantzintla, Puebla.

[27] Sossa, H., Barrón, R., Vazquez, R. *RealValued Pattern Classification based on Extended Associative Memory*. En Proceedings of the Fifth Mexican International Conference in Computer Science, 2004. ENC 2004., Colima, Mexico, 2004, pp. 213-219.

[28] Steinbuch, K. *Die Lernmatrix*. En Kybernetik, 1(1), pp 26-45

[29] Barron, R. *Associative Memories and Morphological Neural Networks for Patterns Recall*. PhD dissertation, Center for Computing Research - National Polytechnic Institute. México, 2006.

[30] Abduo, M., Galster, M. *Myo Gesture Control Armband for Medical Applications*. Departamento de Ciencias de la computación e Ingeniería en Software, Universidad de Canterbury, 2015.

[31] Pololu Robotics & Electronics *150:1 Micro Metal Gearmotor HPCB 6V with Extended Motor Shaft* . Obtenido de: https://www.pololu.com/product/3076 Consultado en junio de 2018.

[32] Pololu Robotics & Electronics *Pololu Dual MC33926 Motor Driver Shield for Arduino* . Obtenido de: https://www.pololu.com/product/2503 Consultado en junio de 2018.

[33] Pololu Robotics & Electronics *Magnetic Encoder Pair Kit for Micro Metal Gearmotors, 12 CPR, 2.7-18V (HPCB compatible)* . Obtenido de: https://www.pololu.com/product/3081 Consultado en junio de 2018.

[34] Myo developer blog *Big Data: Raw EMG Free for Developers in December*. Obtenido de: http://developerblog.myo.com/big-data/ Consultado en Enero de 2018.

[35] Thalmic Labs *Brazalete MYO*. Obtenido de: https://developer.thalmic.com/ Consultado en Enero de 2018.

[36] González, C. *Aplicaciones orientadas a la domótica con Raspberry Pi*. Trabajo Fin de Grado en Ingeniería de las Tecnologías de Telecomunicación, Universidad de Sevilla, 2015.

[37] Raspberry Pi *Raspberry Pi 3 model B*. Obtenido de: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ Consultado en Enero de 2018.

[38] GitHub *dzhu/myo-raw*. Obtenido de: https://github.com/dzhu/myo-raw Consultado en Enero de 2018.

[39] Gómez, J. *Las prótesis: restauración del individuo*. Revista Ciencia y Desarrollo, Vol. 32, Núm. 196, pp 62-67; Consejo Nacional de Ciencia y Tecnologia, México, Junio 2006.