Desarrollo de un sistema de control para el movimiento de un prototipo de prótesis de mano a partir del reconocimiento de señales Mioeléctricas

Diego A. Ballén Garnica, Marco A. Velasco, Jesús Villarreal López

Universidad Santo Tomas, Facultad de Ingeniería Mecánica, Carrera 9 51-11, Bogotá, Colombia

Resumen

En algunos países, debido al alto costo que involucran las soluciones protésicas por su sistema de control, las personas con amputaciones tienen un acceso limitado a éstas, presentándose así, un crecimiento significativo de dificultades para que las personas vivan una vida activa. De allí que, a través de este trabajo, se muestra un sistema de control con base en un desarrollo mecánico y eléctrico para la implementación de un prototipo de prótesis de mano compuesta por un microcontrolador (Raspberry Pi), un sensor de señales mioeléctricas (requerido para la detección de la diferencia de potencial generado por la contracción muscular) y actuadores eléctricos (para el movimiento de los dedos del sistema protésico). El prototipo de prótesis fabricado para favorecer la viabilidad del sistema de control elaborado, consiste en una mano de cinco dedos manufacturada por la técnica de Modelado por deposición fundida (FDM). La factibilidad del proyecto se evidencia en el análisis del desempeño de la prótesis con el sistema de control sugerido, por medio del estudio de las variables de movimientos predeterminados.

Palabras clave: electromiografía, prótesis, impresión 3D.

1 Introducción

Se ha observado en los últimos años un crecimiento considerable en la tasa de porcentaje de personas con amputaciones de una de sus extremidades superiores, dificultándose así el mejoramiento de su calidad de vida, debido a las limitaciones en el desarrollo de las actividades cotidianas (Departamento Administrativo Nacional de Estadística, 2004). En todo el mundo, cada año, el número de amputados aumenta entre 150,000 y 200,000 nuevos casos, que se agregan a los 4 millones existentes. El 30% de estos amputados han sufrido una amputación de miembro superior, de los cuales el 60% de las amputaciones de brazo se encuentran en personas en un rango de edad entre 21 y 64 años de edad; mientras que un 10% son pacientes menores de 21 años (Toledo, Leija, Muñoz, Vera, & Ramírez, 2009). Teniendo en cuenta tal situación, se han desarrollado sistemas protésicos para superar de una y otra forma las restricciones físicas debidas a las amputaciones.

En los últimos años se han construido prótesis con características mecánicas, eléctricas e híbridas, las mecánicas se caracterizan por ser dispositivos que utilizan un sistema único de apertura o cierre voluntario por medio de un arnés controlado por el usuario; sin embargo, estas se limitan al agarre de objetos relativamente grandes (Manuel & González, 2004). Otras son las prótesis electrónicas, las cuales utilizan varios motores eléctricos, con una batería recargable y que es posible controlarlas de varias formas, ejemplo: un servo control, un botón pulsador o un interruptor con arnés, etc. (Díaz, 2016). Las prótesis electrónicas frente a las prótesis mecánicas permiten mayor movilidad tanto en los dedos como en el brazo, esto hace que el costo de esta aumente a diferencia de las prótesis mecánicas. Además, se han desarrollado prótesis hibridas que consisten en combinar el movimiento del cuerpo con el accionamiento por electricidad. Este concepto es ampliamente utilizado en las

prótesis transhumerales (amputación por encima del codo), donde por lo general el codo es accionado por el cuerpo y el dispositivo terminal (gancho o mano) es de accionamiento eléctrico (Loaiza & Arzola, 2011).

Al observar el desarrollo y la evolución de los sistemas protésicos, cabe resaltar algunos trabajos que se enfocaron a encontrar soluciones factibles donde el sistema protésico mejore la calidad de vida de los amputados. Por ejemplo, el estudio realizado por la Universidad Autónoma de México denominado "Control de modelo de prótesis de mano por señal mioeléctrica" se basó en el tratamiento de señales a través de un control desde un software Labview para el accionamiento de la prótesis (Armas-Álvarez, 2017). Por su parte, la Universidad Militar Nueva Granada de Colombia desarrolló el proyecto "Control mioeléctrico de una prótesis de miembro superior - mano" logrando captar las señales y aplicar un sistema de control con el software Matlab implementando redes neuronales para aumentar el número de movimientos de la prótesis (Arias, 2017). También se destaca el proyecto desarrollado en la Universidad de Valladolid de España "Entrenador mioeléctrico de prótesis para amputados de brazo y mano" el cual tuvo como objetivos crear un entrenador mioeléctrico virtual para evaluar la adecuación de una prótesis de señales mioeléctricas para amputados de mano y posibilitar un aprendizaje previo de manejo de prótesis (Alonso Alonso, Hornero Sánchez, Espino Hurtado, de la Rosa Steinz, & Liptak, 2002). Se aprecia entonces que el control de prótesis de miembro superior es un tema en constante desarrollo pero que requiere el uso de software y hardware que ayude a disminuir el peso y al avance tecnológico de la solución propuesta.

Por lo tanto, el presente artículo tiene por finalidad presentar el desarrollo de un sistema de control para el movimiento de un prototipo de una prótesis de mano a partir del reconocimiento de señales mioeléctricas. El prototipo desarrollado se basa en un sensor Myo armband, una placa controladora Raspberry y programación básica en Python.

2 Materiales y Métodos

Para la implementación del sistema de control mioeléctrico, se desarrolló un sistema de instrumentación, un sistema de control, un sistema de actuadores y un sistema electromecánico.

2.1 Sistema de instrumentación

2.1.1 Myogesture armband

En este trabajo, se usa un sensor mioeléctrico conocido como Myo Armband (Thalmic laboratories, Canadá), este posee 8 electrodos de acero inoxidable para la lectura de señales electromiográficas, los cuales se encargan del reconocimiento y la caracterización de 5 gestos predeterminados (Duble_Tap, Fingers_Spread, Wave_Out, Wave_In, Fist) (Figura 1) (Sathiyanarayanan & Rajan, 2016). Este cuenta con giroscopio, acelerómetro y tres ejes para posicionamiento, además este sensor a través de una retroalimentación háptica permite saber el comportamiento que está desarrollando el sensor a trascurso del tiempo por medio de vibraciones largas medias o cortas (NORTH, 2012).

Figura 1. Gestos predefinidos del sensor MYO. Fuente: Myo

2.1.2 Adquisición de datos

El sensor Myo armband hace uso de la electromiografía (EMG) que es la técnica que registra la actividad eléctrica relacionada con el funcionamiento de los músculos esqueléticos. La detección se puede realizar por medio de electrodos intramusculares los cuales se exponen directamente a la señal del musculo o superficiales los cuales se ponen sobre la piel (Fernández, Acevedo, & Tabernig, 2007). La toma de señales electromiográficas depende del número de electrodos que se estén usando, generando así la clasificación de los diferentes tipos de sensores superficiales que existen, como lo son: sensor monopolar (con un único electrodo activo), sensor bipolar (con dos electrodos activos) y sensor multi-electrodo (con 2 o más electrodos) (Moreno, 2011). El sensor Myo armband es del tipo de multielectrodo superficial y para la captación de señales fue conectado en el antebrazo según la posición sugerida por el fabricante (Figura 2).

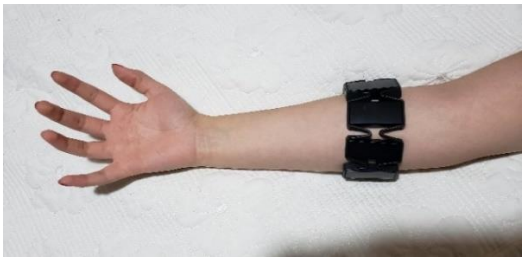


Figura 2. Posición del sensor en el brazo. Fuente: Autor

Al realizar la adquisición de datos del sensor, las señales se capturan a una velocidad de 200 Hz que es la estándar para el sensor. Cada vez que se conecta el brazalete, se crean conjuntos de archivo de registro que ayudarán al programa de control para la ejecución de gestos (Bernhardt, 2015).

El sensor cuenta con unas salidas clasificadoras las cuales son procesadas por algoritmos del software de adquisición de datos. Es decir, el sensor no muestra la señal en bruto directamente si no la transforma a una señal establecida. Esto se debe a que las señales electromiográficas de dos personas diferentes (A y B) para realizar un mismo gesto son bastante diferentes como se ve en la Figura 3. (Steve, 2014).

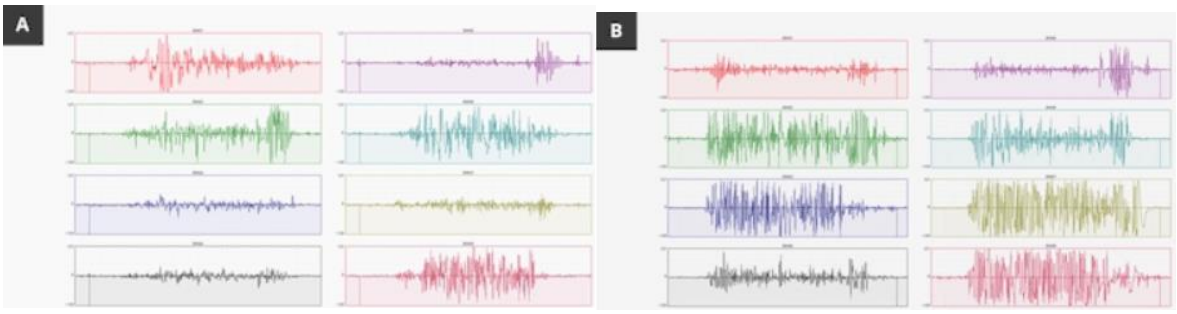


Figura 3. Señales electromiográficas para un mismo gesto A) Prueba de señales del sujeto B) Prueba de señales del sujeto B. Nótese la diferencia de forma y amplitud de señal para cada uno de los sensores. Fuente: Thalmic Laboratories (Steve, 2014)

Por otro lado, con el fin de realizar la lectura de las señales adquiridas, se usó una placa controladora para la adquisición y manipulación de datos necesarios. La placa utilizada es una Raspberry Pi 3 Modelo B + (Fundación Raspberry PI, Reino Unido), la cual trabaja con un procesador de cuatro núcleos de 64 bits que funciona a 1,4 GHz, con comunicación LAN inalámbrica de doble banda de 2,4 GHz y 5 GHz y Bluetooth 4.2 / BLE (Upton, 2018) (Figura 7). Ésta placa se eligió por sus capacidades para el procesamiento de datos, funcionalidad y tamaño en comparación con otras placas lo que posibilita que la prótesis tenga funcionalidad inalámbrica e independiente.

2.1.3 Transmisión de los datos

Para la transmisión y el procesamiento de datos, se utilizaron dos protocolos. El primer protocolo es el UART (Universal Asynchronous Reciver/Transmitter, UART), que convierte los datos tipo serie a datos tipo paralelo cuando se trata de datos de entrada y convierte datos de tipo paralelo a datos de tipo serie para la transmisión (datos de salida) (Marín, 2005) lo que permite el uso del adaptador Bluetooth que tiene el sensor integrado (Boed, 1999). Este adaptador se comunica con el que se encuentra conectado a través de puerto USB en la placa controladora Raspberry.

2.2 Sistema de control

Una vez adquiridas las señales electromiográficas es necesario determinar el sistema adecuado para lograr el control de las posiciones de los motores. Dentro de los posibles sistemas de control se encuentran: sistema de encendido/apagado, control proporcional, directo, control de máquina de estado finito, patrón reconocimiento, postura de control y control de regresión (Geethanjali, 2016). Luego de realizar comparaciones de aplicabilidad, se determinó usar el sistema de control por máquina de estado finito, puesto que esta técnica permite la clasificación de posturas determinándolas en estados predefinidos (Figura 4) (Cipriani, Zaccone, Micera, & Carrozza, 2008).

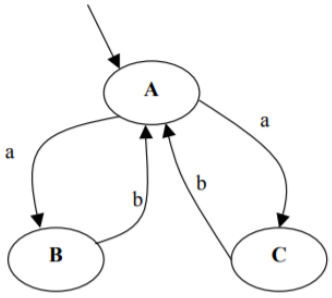


Figura 4. Sistema de control FSM Fuente: (Alavi, Aliaga, & Murga, 2016)

Para la obtención de las señales del sensor y la comunicación con la placa controladora, se usó una secuencia de códigos del programa IDLE de Python 2.7 (Python Software Foundation, Holanda), el SDK de Myo armband, la adaptación de Dhzu para la conexión de Python con Myo Armband y la adaptación para el Sistema operativo de raspberry del Equipo Brink.IO (Harshavardhan, Mohan, & Praveen, 2018) (Figura 5). Los códigos tienen una escritura sencilla basada en código C que facilita la interpretación del sistema de control para la ejecución de la lectura de las salidas del sensor y

determinar el tipo de señales tomadas (Figura 5). El código de programación en Python realiza un procesamiento de datos teniendo en cuenta variables, tales como: como el número de gestos, tiempo de reacción, ángulos de movimiento; para así poder determinar el tipo de señal recibido y así generar datos de salida (Ver anexo).

```
import sys
import time
sys.path.append ('../lib/')

from myo import Myo
from print_pose_listener import PrintPoseListener
from vibration_type import VibrationType
from pose_type import PoseType
from device_listener import DeviceListener
def main():
    print('Start Myo for Linux')

    listener = PrintPoseListener()
    myo = Myo()

    try:
        myo.connect()
        myo.add_listener(listener)
        myo.vibrate(VibrationType.SHORT)
        while True:
            myo.run()

    except KeyboardInterrupt:
        pass
    except ValueError as ex:
        print(ex)
    finally:
        myo.safely_disconnect()
        print('Finished.')
if __name__ == '__main__':
    main()
```

Figura 5. Programación para ejecución de Myo en Python Fuente: Autor

Para la configuración complementaria de la placa, se crearon códigos base, ya que sin éstos el programa no se ejecutaría en el entorno del desarrollo y aprendizaje integrado de Python. Adicional a ello, se organizan códigos en la terminal para crear una aplicación de ejecución con el fin de acceder a la configuración base de la placa, permitiendo así el encendido de la prótesis por parte del usuario. Esto se logra por la configuración en el editor Raspbian a través de los códigos escritos en Sudo Nano que es un editor de texto sencillo que se usa en la Terminal (UNAM, 2005). Dicho código puede ser utilizado como un comando de reinicio y ejecución automática del control de movimiento de la prótesis.

2.3 Sistema de actuadores

El sistema de actuadores se hizo para controlar el sistema mecánico desarrollado a partir del proyecto Inmoov (Langevin, 2014) y del trabajo de Bermeo et al. (BERMEO, 2018) y que se describe en detalle en la sección 2.4.1 de este artículo. El sistema mecánico consta de cuatro dedos con capacidad de extensión-flexión y un quinto dedo pulgar con capacidad de extensión, flexión y oposición. Este fue realizado mediante manufactura aditiva de modelado por deposición fundida (FDM) usando una impresora Zortrax M200 (Zortrax, Polonia). El material utilizado es Z-Ultrat, 90% ABS y 10% de

aditivos, con características de durabilidad y adaptabilidad para prototipos funcionales (Zortrax, 2019). La impresión del prototipo se usó una configuración de espesor de capa de 0.1 mm y relleno panal con una densidad del 70% para las piezas de alta resistencia (BERMEO, 2018).

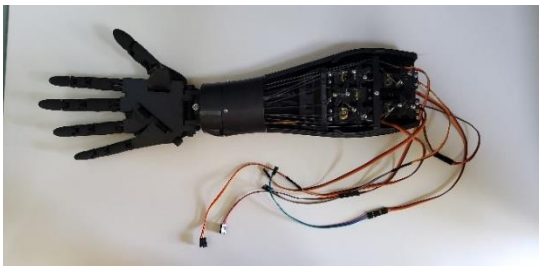


Figura 6. Prótesis realizada. Fuente: Autor

Para el movimiento de los dedos se elaboró un sistema de cables que conectara a los efectores con sus respectivos actuadores. Los actuadores usados son cinco servos MG995 (Tekscan™, Estados Unidos) que se seleccionaron considerando criterios de selección como el tamaño, precio, durabilidad, fuerza y peso (Tabla 1).

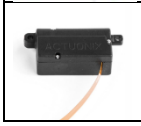
Servomotor	Referencia	Torque (N*M)	Tamaño	Peso (g)	Durabilidad	Precio COP
	MG995	0,98	Largo 40.7 mm, ancho 19.7 mm, altura 42.9 mm	55	Alta durabilidad por sus engranajes metálicos	$ 20.500
	SG90	0,24	Largo 23 mm, ancho 12 mm, altura 32 mm	15	Baja durabilidad por sus engranajes de plástico	$ 6.000
	Actuador Lineal PQ12 30:1	Fuerza de 15 N	Largo 36.5 mm, ancho 21.5 mm, altura 68 mm	15	Media durabilidad por sus engranajes metálicos y estructura de plástico	$ 231.000

Tabla 1. Selección de actuadores Fuete: Autor

El sistema de actuadores se comunica con la placa Raspberry a través de a una conexión de protocolo Rx/Tx, la cual transmite y recibe los datos a una placa SSC-32 servo controller (Robotshop, Canadá) que tiene por objeto la lectura de datos de movimiento y lleva a la ejecución final del sistema protésico.

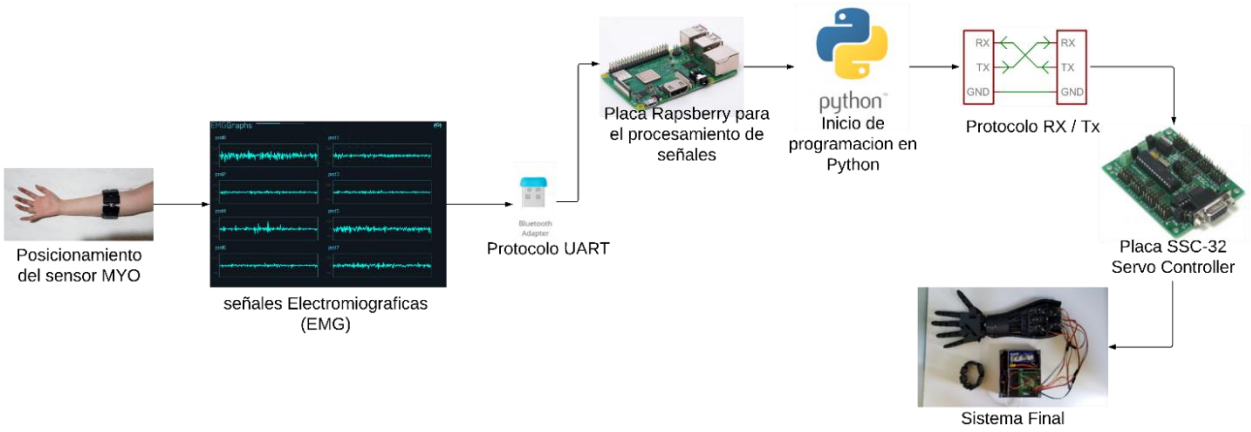


Figura 7. Pasos para el desarrollo del proyecto Fuente: Autor

2.4 Sistema electromecánico.

2.4.1 Sistema Mecánico

El sistema mecánico tiene por base la anatomía del cuerpo humano (en este caso la mano), los movimientos y ángulos que podría realizar cada uno de los dedos. Por lo tanto, el sistema, al igual que una mano natural, cuenta con catorce falanges distribuidas en el pulgar que posee dos falanges y los otros cuatro dedos tres falanges. Las articulaciones interfalángicas son articulaciones tipo bisagra que permiten flexión y extensión. Para lograr estos movimientos se cuenta con tendones flexores encargados de realizar el movimiento. Estos son remplazados en la prótesis por cables o cuerdas que se encargan de realizar el movimiento requerido al momento de hacer alguna acción (Figuras 8 y 9) (Cailliet, 2006). Para el control de los movimientos se tuvieron en cuenta los ángulos de flexión y extensión máximos que se poseen las distintas articulaciones de la prótesis (Tabla 2) (Cobos, Ferre, Sanchéz-Urán, Ortego, & Peña, 2008).

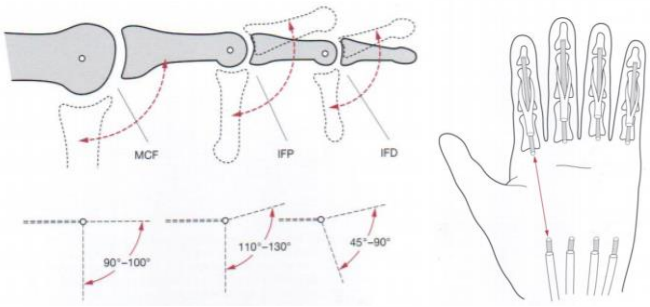


Figura 8. Falanges y tendón de la mano Humana Fuente: Cailliet

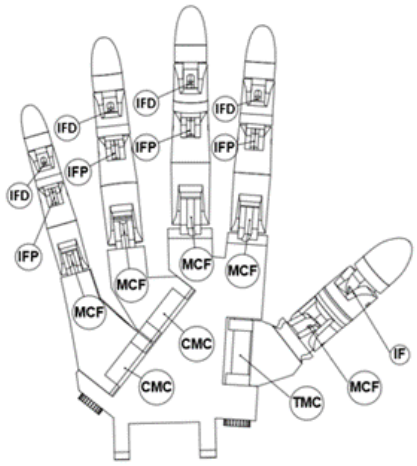


Figura 9. Disposición para el análisis dinámico de la prótesis Fuente: Bermeo

ARTICULACIÓN	FLEXIÓN	EXTENSIÓN	ABDUCCIÓN - ADUCCIÓN
Pulgar			
Trapeciometacarpiana (TMC)	50º - 90º	15º	45º-60º
Metacarpofalángica (MCF)	75º - 80º	0º	0º
Interfalángica (IF)	75º - 80º	5º - 10º	0º
Índice			
Carpo-Metacarpal (CMC)	5º	0º	0º
Metacarpofalángica (MCF)	90º	30º - 40º	60º
Interfalángica Proximal (IFP)	110º	0º	0º
Interfalángica Distal (IFD)	80º - 90º	5º	0º
Medio			
Carpo-Metacarpal (CMC)	5º	0º	0º
Metacarpofalángica (MCF)	90º	30º - 40º	60º
Interfalángica Proximal (IFP)	110º	0º	0º
Interfalángica Distal (IFD)	80º - 90º	5º	0º
Anular			
Carpo-Metacarpal (CMC)	10º	0º	0º
Metacarpofalángica (MCF)	90º	30º - 40º	45º
Interfalángica Proximal (IFP)	120º	0º	0º
Interfalángica Distal (IFD)	80º - 90º	5º	0º
Meñique			
Carpo-Metacarpal (CMC)	15º	0º	0º
Metacarpofalángica (MCF)	90º	30º - 40º	50º
Interfalángica Proximal (IFP)	135º	0º	0º
Interfalángica Distal (IFD)	90º	5º	0º

Tabla 2. Ángulos máximos de Flexión Y Extensión Fuente: Cobos

2.4.2 Sistema Eléctrico

El sistema eléctrico cuenta con una fuente de alimentación de circuito regulador para una doble reducción de voltaje 11.1 V a 5 V y 6 V. Al usar una Batería Lipo Turnigy 2200mAh 11.1V como fuente principal, se implementó el uso de dos placas LM2596 de regulación (Texas Instruments, Estados Unidos) capaces de manejar una carga de hasta 3 A (Description, Application, Diagrams, & Information, 1996). Estos dispositivos alimentan con 5V la placa Raspberry la otra con 6V que alimenta la placa controladora SSC32 Proteus (Figura 10).

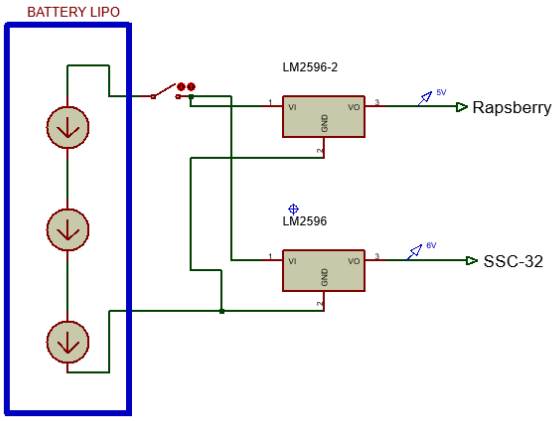


Figura 10. Circuito para regulación de voltaje. Fuente: Autor

Una vez definidos el sistema eléctrico y de control se fabricó un elemento de almacenamiento para que todos sistemas estén ubicados en un solo lugar (Figura 11). La caja fue construida en resina Grey a través del sistema de impresión estereolitográfica (SLA) con una impresora Form2 (Formlabs, Estados Unidos) la cual funciona mediante la exposición de una capa de resina líquida fotosensible a un rayo láser UV para que se endurezca y se solidifique (Locker, n.d.).

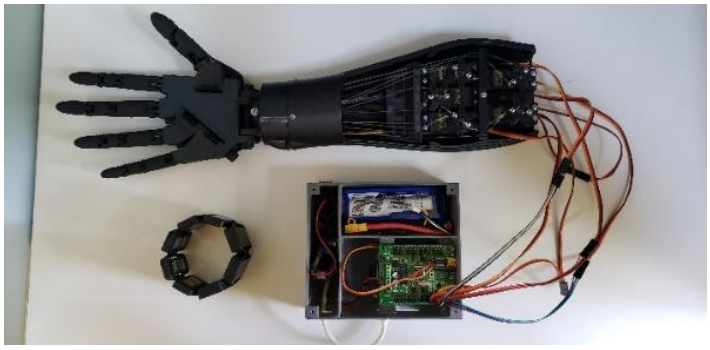


Figura 11. Sistema electromecánico. Fuente: Autor

2.5 Resultados

Como se ha podido observar a lo largo del artículo, se ha logrado la implementación de un sistema de control de estado finito (Figura 12) el cual permite la ejecución de movimientos de la prótesis. Este sistema de control funciona mediante la detección de los cinco gestos predefinidos del sensor Myo Armband (Fingers_Spread, Duble_Tap, Fist, Wave_out, Wave_in), que una vez identificados y procesados permiten generar cinco movimientos (apertura total, tres dedos, cierre total, señalar, pulgar) (Tabla 4). Dichos movimientos en el sistema de control, se interpretan de la siguiente manera: los círculos que están distribuidos en el diagrama son los resultados de los movimientos obtenidos, los cuales son llamados como S1, S2, S3, S4, S5 cada movimiento final responde a las señales del brazalete por lo tanto se puede determinar que la lectura de los datos va a estar saltando de pose a pose hasta llegar al movimiento establecido previamente en señales nombradas como e1, e2, e3, e4, e5 (Tabla 3). Por ejemplo, si se desea lograr el movimiento de apertura total, la señal programada para la ejecución de este movimiento es Fingers_spread por lo tanto, el Sistema de Control tiene que saltar de pose a pose hasta llegar a S1 y hacer la verificación del movimiento establecido.

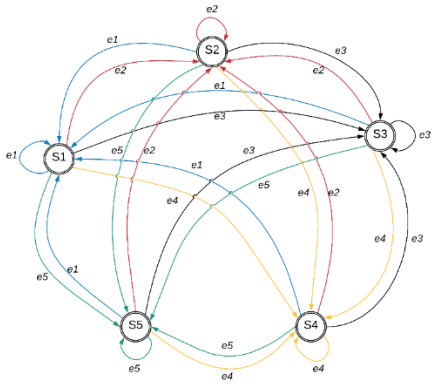
Estado	Acciones	Estado	Resultado	Gráfica de estados finitos
e1	Fingers_Spread	S1	Apertura total	
e2	Duble_Tap	S2	Tres dedos	
e3	Fist	S3	Cierre total	
e4	Wave_out	S4	Señalar	
e5	Wave_in	S5	Pulgar	

Tabla 3. Acciones del Sistema de control de máquina de estado finito Fuente: Autor

Para evaluar la funcionalidad del sistema de control, se evalúan los cinco movimientos descritos en la Tabla 3. Estos cinco gestos son los espacios preestablecidos que puede detectar el sensor y que se consideran en el sistema de control de máquina de estado finito. En las pruebas realizadas se pudo observar que en la lectura de los datos correspondientes a los dedos meñique, anular y dedo medio, el comportamiento de las señales electromiográficas tienen un potencial eléctrico similar, haciendo que la lectura de estas, sean erróneas y pueda manifestarse una mala lectura de gestos deseados

Nombre del gesto	Resultado del Gesto	Tiempo de reacción (s)	Porcentaje de acierto	Imagen del gesto	Resultado del sistema
Fingers_Spread	Apertura Total	3.638	53.3%		

Duble_Tap	Tres dedos	2.157	80%		
Fist	Cierre Total	1.327	93.3%		
Wave_out	Señalar	1.924	60%		
Wave_in	Pulgar	1.759	66.6%		

Tabla 4. Tiempos de reacción para ejecución de movimientos. Fuente: Autor

Gracias al sistema de control, se logra desarrollar un sistema de control que como ventaja principal tiene un menor número de componentes, tamaño y peso en comparación con el trabajo realizado previamente (Figura 13) (BERMEO, 2018). Esto se logró con una placa controladora Raspberry en la que se implementa un sistema de control de máquina de estado finito en Python (Figura 14). Sin embargo, frente al sistema de control anterior que estaba soportado en computador personal, placa Arduino y Matlab tiene un problema de saturación, debido a que la placa Raspberry debe procesar y transmitir datos de forma simultánea con una menor capacidad de cómputo. Se puede observar que el tiempo de reacción entre la detección de las señales electromiográficas y la ejecución del movimiento en la prótesis varia de 1.5 a 4 segundos dependiendo del gesto (Tabla 4), también se realizan quince pruebas por cada gesto y así definir un porcentaje de aciertos para determinar cuál gesto causa más inconvenientes a la hora de la ejecución.

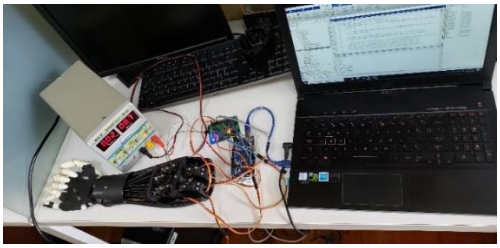


Figura 12. Proyecto previo realizado en Matlab Fuente: Bermeo

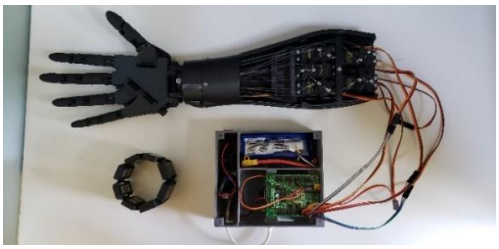


Figura 13. Proyecto actual realizado en Python Fuente: Autor

3 Conclusiones

- Respecto a las señales EMG se logró observar las magnitudes que presentan y su variabilidad, dependiendo del sujeto de prueba y su condición física. Se conoce que las señales dependen del esfuerzo aplicado al músculo, sin dejar de lado los distintos tipos de fatiga, ya que éstos hacen que las señales se debiliten. Por lo tanto, para la generación de estrategias de medición de las señales mioeléctricas, es necesario tener un conocimiento básico sobre la actividad muscular que se desea estudiar teniendo en cuenta que, en el caso de los dedos índice, medio y anular, se ven afectados por el mismo músculo, haciendo que las características estudiadas presentaran una tendencia similar.

- Se desarrolló un sistema de control mioeléctrico, el cual demuestra que se puede lograr una prótesis con alta funcionalidad con elementos asequibles que existen en el mercado como lo son: servomotores, placas controladoras, baterías, etc. obteniendo varias ventajas para este sistema protésico, como portabilidad, un encendido sencillo y un control de los gestos predefinidos para el movimiento. Adicional a ello, se posee un Sistema de Control reprogramable con acceso a la placa Raspberry y un conocimiento de programación en Python.

- Para el caso del uso de estrategias de control basadas en máquinas de estado finito, se concluye que dicha estrategia es útil cuando se posee un número fijo de posiciones identificadas de manera adecuada; pero carece de utilidad cuando es necesario establecer múltiples funciones a partir de posiciones establecidas, debido a que el cambio de estado se produce desde la detección de la señal EMG hasta que la posición es identificada y relacionada como variable válida.

- Se logró observar y comparar los sistemas de control desarrollados en Matlab y Python con diferente hardware y así tener claro el comportamiento que tienen respecto al procesamiento de los datos y funcionalidad de la prótesis, concluyendo que, el sistema de Matlab en conjunto con un computador es mejor para el procesamiento de datos a diferencia de Python y Raspberry por su tiempo de reacción; pero, si se tiene en cuenta el número de dispositivos que usa cada sistema podemos observar que el proyecto logro un avance para dar portabilidad

para uso del usuario y así acercarse más a un sistema protésico que pueda ser usado en la vida real.

4 Recomendaciones

Para futuros trabajos y desarrollo de este proyecto, se tienen varias recomendaciones las cuales son:

1. Trabajar con un diseño de prótesis adaptable al cuerpo humano aligerando cargas, tamaño y peso, ya que esta prótesis actual trabaja con una prótesis para realizar experimentos de funcionalidad y su peso puede ser aligerado.
2. Estudiar el comportamiento de una prótesis flexible que es otro tipo de prótesis interesante para trabajar pues disminuiría la cantidad de partes móviles.
3. Continuar con la disminución de tamaño y peso del sistema de control para que los elementos electrónicos y eléctricos estén todos posicionados dentro de la prótesis.
4. Implementar un sistema de control de reconocimiento de patrones, ya que al lograr implementar las ecuaciones necesarias y el entrenamiento se podría llegar a un comportamiento más fluido y natural de una mano protésica.
5. Estudiar otro tipo de sensores mioeléctricos que permitan ampliar las posibilidades de trabajo a las que se están limitadas cuando se usa la Myo Armband.

5 Referencias

Alavi, M., Aliaga, S., & Murga, M. (2016). Máquinas de Estado Finito. *Iluminate*, *8*(1), 41–57. Retrieved from http://www.revistasbolivianas.org.bo/pdf/riei/v8n1/v8n1_a05.pdf

Alonso Alonso, A., Hornero Sánchez, R., Espino Hurtado, P., de la Rosa Steinz, R., & Liptak, L. (2002). Entrenador mioeléctrico de prótesis para amputados de brazo y mano [Myolectric prostheses trainer for hand and arm amputees]. *Mapfre Medicina*, *13*(1), 11–19. Retrieved from http://sid.usal.es/5924/8-2-6

Arias, N. A. (2017). *Control Mioeléctrico De Una Prótesis De Miembro Superior - Mano*. UNIVERSIDAD MILITAR NUEVA GRANADA.

Armas-Álvarez, A. E. (2017). Control de modelo de prótesis de mano por señal mioeléctrica. *Memorias Del Congreso Nacional de Ingeniería Biomédica*, (1), 328–331. Retrieved from http://memorias.somib.org.mx/index.php/memorias/article/view/125

BERMEO, J. (2018). *DESARROLLO DE UN SISTEMA DE CONTROL POR MEDIO DE SENSORES MIOELÉCTRICOS PARA UNA PRÓTESIS DE BRAZO TRANSRADIAL*. Universidad Santo Tomas.

Bernhardt, P. (2015). Myo | #MyoCraft: Logging IMU and Raw EMG Data. Retrieved November 28, 2019, from https://developerblog.myo.com/myocraft-logging-imu-and-raw-emg-data/

Boed, V. (1999). Serial communications. *Networking and Integration of Facilities Automation Systems*, (Chapter 10), 69–80. https://doi.org/10.1201/9781420050523-5

Cailliet, R. (2006). *Anatomia Funcional, Biomecanica* (1st ed.). Marban.

Cipriani, C., Zaccone, F., Micera, S., & Carrozza, M. C. (2008). On the shared control of an EMG-controlled prosthetic hand: Analysis of user-prosthesis interaction. *IEEE Transactions on Robotics*, *24*(1), 170–184. https://doi.org/10.1109/TRO.2007.910708

Cobos, S., Ferre, M., Sanchéz-Urán, M. A., Ortego, J., & Peña, C. (2008). Efficient human hand kinematics for manipulation tasks. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*, (October), 2246–2251. https://doi.org/10.1109/IROS.2008.4651053

Departamento Administrativo Nacional de Estadística. (2004). Información estadística de la discapacidad. *Dane*, 41. https://doi.org/10.1016/j.medcli.2009.09.047

Description, G., Application, T., Diagrams, C., & Information, O. (1996). *LM2594 SIMPLE SWITCHER Power Converter 150 kHz 0 5A Step-Down Voltage Regulator SIMPLE SWITCHER Power Converter 150 kHz 0 5A Step-Down Voltage Regulator LM2594-12*. (July 1995).

Díaz, leidy. (2016). Prótesis electrónicas: Una nueva esperanza para mejorar la calidad de vida de las personas. *Journal of Chemical Information and Modeling*, *53*(9), 169–174. https://doi.org/10.1017/CBO9781107415324.004

Fernández, J., Acevedo, R., & Tabernig, C. (2007). Influencia de la fatiga muscular en la señal electromiográfica de músculos estimulados eléctricamente. *Revista EIA*, (7), 111–119. https://doi.org/10.24050/reia.v4i7.175

Geethanjali, P. (2016). Myoelectric control of prosthetic hands: State-of-the-art review. *Medical Devices: Evidence and Research*, *9*, 247–255. https://doi.org/10.2147/MDER.S91102

Harshavardhan, Mohan, K., & Praveen, B. (2018). Rover controlado con Myo Armband y Raspberry Pi 3 - Hackster.io. Retrieved January 15, 2020, from https://www.hackster.io/brink-io/rover-controlled-using-myo-armband-and-raspberry-pi-3-c09379

Langevin, G. (2014). Inmoov Hand. Retrieved from http://inmoov.fr/hand-and-forarm/

Loaiza, J. L., & Arzola, N. (2011). Evolución Y Tendencias En El Desarrollo De Prótesis De Mano Evolution and Trends in the Development of Hand Prosthesis. *Dyna*, *78*, 191–200. Retrieved from http://www.scielo.org.co/pdf/dyna/v78n169/a22v78n169.pdf

Locker, A. (n.d.). 9 tipos de impresoras 3D: guía de tecnologías de impresión 3D. Retrieved from https://all3dp.com/es/1/tipos-de-impresoras-3d-tecnologia-de-impresion-3d/

Manuel, J., & González, D. (2004). *ROBÓTICA Y PRÓTESIS INTELIGENTES*. 1–15.

Marín, M. (2005). *Tema 6. Comunicaciones asíncronas (UART's)*. 10. Retrieved from http://www.el.uma.es/marin/Practica4_UART.pdf

Moreno, I. (2011). *Estudio electromiográfico de los patrones musculares en sujetos con mordida abierta anterior*. https://doi.org/10.1016/S0141-0229(03)00220-5.L

NORTH. (2012). Especificaciones técnicas de Myo Gesture Control Armband - Bienvenido a Myo Support. Retrieved November 27, 2019, from https://support.getmyo.com/hc/en-us/articles/202648103-Myo-Gesture-Control-Armband-tech-specs

Sathiyanarayanan, M., & Rajan, S. (2016). MYO Armband for physiotherapy healthcare: A case study using gesture recognition application. *2016 8th International Conference on Communication Systems and Networks, COMSNETS 2016*, (January). https://doi.org/10.1109/COMSNETS.2016.7439933

Steve, M. A. (2014). Myo | Announcing Raw EMG Data for Developers from the Myo Armband. Retrieved November 28, 2019, from https://developerblog.myo.com/big-data/

Toledo, C., Leija, L., Muñoz, R., Vera, A., & Ramírez, A. (2009). Upper limb prostheses for amputations above elbow: A review. *2009 Pan American Health Care Exchanges - PAHCE 2009*, (December 2013), 104–108. https://doi.org/10.1109/PAHCE.2009.5158375

UNAM. (2005). El editor nano. Retrieved from http://linux.iingen.unam.mx/pub/Documentacion/Editores-Linux/

Upton, E. (2018). *Raspberry Pi 3 Model B, url:https://www.raspberrypi.org/blog/raspberry-pi-3-model-bplus-sale-now-35/*. 2837. Retrieved from https://www.raspberrypi.org/blog/raspberry-pi-3-model-bplus-sale-now-35/

Zortrax. (2019). Z-ULTRAT. Retrieved from https://store.zortrax.com/es/materials/m200-z-ultrat

6 Anexos

6.1 Librerías usadas

6.1.1 Librería Pose_type

```
#configuración para el control de la pose convirtiendo el tipo de pose en variables numéricas

from enum import Enum

class PoseType(Enum):
        REST = 0
        FIST = 1
        WAVE_IN = 2
        WAVE_OUT = 3
        FINGERS_SPREAD = 4
        DOUBLE_TAP = 5
        UNKNOWN = 255
```

6.1.2 Librería Vibration_type

```
# configuración para el control de la retroalimentación haptica convirtiendo el tiempo de vibración en variables numéricas

from enum import Enum

class VibrationType(Enum):
        NONE = 0
        SHORT = 1
        MEDIUM = 2
        LONG = 3
```

6.1.3 Librería Utilities

```
#Definición de argumentos y estructuras del sensor

import struct

def pack(fmt, *args):
        return struct.pack('<' + fmt, *args)
```

```
def unpack(fmt, *args):
	return struct.unpack('<' + fmt, *args)

def multichr(values):
	return ''.join(map(chr, values))

def multiord(values):
	return map(ord, values)
```

6.1.4 Librería Device_listener

```
from utilities import *

class DeviceListener(object):
	def handle_data(self, data):
		if data.cls != 4 and data.command != 5:
			return

		connection, attribute, data_type = unpack('BHB', data.payload[:4])
		payload = data.payload[5:]

		if attribute == 0x23:
			data_type, value, address, _, _, _ = unpack('6B', payload)

			if data_type == 3:
				self.on_pose(value)

	def on_pose(self, pose):
		pass
```

6.1.5 Librería Packet

```
#paquete de recursos para la configuración del bluetooth

from utilities import *

class Packet(object):
    def __init__(self, ords):
        self.data_type = ords[0]
        self.cls = ords[2]
        self.command = ords[3]
        self.payload = multichr(ords[4:])

    def __repr__(self):
        return 'Packet(%02X, %02X, %02X, [%s])' % \
            (self.data_type, self.cls, self.command,
             ' '.join('%02X' % b for b in multiord(self.payload)))
```

6.1.6 Librería BLE

```
#configuración del sistema de bluetooth y configuración del buffer de datos del sensor Myo Armband
#comunicación del protocolo UART con el software Python

import serial
import time
from packet import Packet
from utilities import *
```

```
class BLE(object):
    def __init__(self, tty_port):
        self.ser = serial.Serial(port=tty_port, baudrate=9600, timeout=1, dsrdtr=1)
        self.buffer = []
        self.listeners = []

    def receive_packet(self, timeout=None):
        start_time = time.time()
        self.ser.timeout = None
        while timeout is None or time.time() < start_time + timeout:
            if timeout is not None: self.ser.timeout = start_time + timeout - time.time()
            x = self.ser.read()
            if not x: return None

            packet = self.process_byte(ord(x))
            if packet:
                if packet.data_type == 0x80:
                    self.notify_event(packet)
                return packet

    def process_byte(self, x):
        if not self.buffer:
            if x in [0x00, 0x80, 0x08, 0x88]:
                self.buffer.append(x)
            return None
        elif len(self.buffer) == 1:
            self.buffer.append(x)
            self.packet_length = 4 + (self.buffer[0] & 0x07) + self.buffer[1]
            return None
        else:
            self.buffer.append(x)

        if self.packet_length and len(self.buffer) == self.packet_length:
            packet = Packet(self.buffer)
            self.buffer = []
            return packet
        return None

    def notify_event(self, p):
        for listener in self.listeners:
            if listener.__class__.__name__ == 'function':
                listener(p)
            else:
                listener.handle_data(p)

    def add_listener(self, listener):
        self.listeners.append(listener)

    def remove_listener(self, listener):
        try: self.listeners.remove(listener)
        except ValueError: pass

    def wait_event(self, cls, command):
        response = [None]
        def valid_packet(packet):
            if packet.cls == cls and packet.command == command:
                response[0] = packet
        self.add_listener(valid_packet)
        while response[0] is None:
            self.receive_packet()
        self.remove_listener(valid_packet)
        return response[0]
```

```
    def connect(self, address):
        return self.send_command(6, 3, pack('6sBHHHH', multichr(address), 0, 6, 6, 64, 0))

    def start_scan(self):
        return self.send_command(6, 2, b'\x01')

    def end_scan(self):
        return self.send_command(6, 4)

    def read_attribute(self, connection, attribute):
        self.send_command(4, 4, pack('BH', connection, attribute))
        return self.wait_event(4, 5)

    def write_attribute(self, connection, attribute, value):
        self.send_command(4, 5, pack('BHB', connection, attribute, len(value)) + value)
        return self.wait_event(4, 1)

    def send_command(self, cls, cmd, payload=b''):
        package = pack('4B', 0, len(payload), cls, cmd) + payload
        self.ser.write(package)

        while True:
            packet = self.receive_packet()

            if packet.data_type == 0:
                return packet

            self. Notify_event(packet)

    def disconnect(self, h):
        return self.send_command(3, 0, pack('B', h))
```

6.1.7 Librería MYO

#configuración del sensor Myo Armband con el software Python teniendo en cuenta el tipo de gesto, retroalimentación Haptica y buffer de datos del bluetooth

```
import re
from ble import BLE
from serial.tools.list_ports import comports
from utilities import *
from vibration_type import VibrationType

class Myo(object):

    def __init__(self):
        self.ble = None
        self.connection = None

    def connect(self, tty_port = None):
        self.safely_disconnect()
        self.find_bluetooth_adapter(tty_port)

        address = self.find_myo_device()
        connection_packet = self.ble.connect(address)
        self.connection = multiord(connection_packet.payload)[-1]
        self.ble.wait_event(3, 0)
        print('Connected.')

        is_fw_valid = self.valid_firmware_version()
```

```
    if is_fw_valid:
        device_name = self.read_attribute(0x03)
        print('Device name: %s' % device_name.payload[5:])
        self.write_attribute(0x1d, b'\x01\x00')
        self.write_attribute(0x24, b'\x02\x00')
        self.initialize()
    else:
        raise ValueError('The firmware version must be v1.x or greater.')

def find_bluetooth_adapter(self, tty_port = None):
    if tty_port is None:
        tty_port = self.find_tty()
    if tty_port is None:
        raise ValueError('Bluetooth adapter not found!')

    self.ble = BLE(tty_port)

def find_tty(self):
    for port in comports():
        if re.search(r'PID=2458:0*1', port[2]):
            return port[0]

    return None

def run(self, timeout=None):
    if self.connection is not None:
        self.ble.receive_packet(timeout)
    else:
        raise ValueError('Myo device not paired.')

def valid_firmware_version(self):
    firmware = self.read_attribute(0x17)
    _, _, _, _, major, minor, patch, build = unpack('BHBBHHHH', firmware.payload)

    print('Firmware version: %d.%d.%d.%d' % (major, minor, patch, build))

    return major > 0

def add_listener(self, listener):
    if self.ble is not None:
        self.ble.add_listener(listener)
    else:
        print('Connect function must be called before adding a listener.')

def vibrate(self, duration):
    cmd = b'\x03\x01'
    if duration == VibrationType.LONG:
        cmd = cmd + b'\x03'
    elif duration == VibrationType.MEDIUM:
        cmd = cmd + b'\x02'
    elif duration == VibrationType.SHORT:
        cmd = cmd + b'\x01'
    else:
        cmd = cmd + b'\x00'

    self.write_attribute(0x19, cmd)

def initialize(self):
    self.write_attribute(0x28, b'\x01\x00')
    self.write_attribute(0x19, b'\x01\x03\x01\x01\x00')
    self.write_attribute(0x19, b'\x01\x03\x01\x01\x01')
```

```
    def find_myo_device(self):
        print('Find Myo device...')
        address = None
        self.ble.start_scan()
        while True:
            packet = self.ble.receive_packet()

            if packet.payload.endswith(b'\x06\x42\x48\x12\x4A\x7F\x2C\x48\x47\xB9\xDE\x04\xA9\x01\x00\x06\xD5'):
                address = list(multiord(packet.payload[2:8]))
                break

        self.ble.end_scan()
        return address

    def write_attribute(self, attribute, value):
        if self.connection is not None:
            self.ble.write_attribute(self.connection, attribute, value)

    def read_attribute(self, attribute):
        if self.connection is not None:
            return self.ble.read_attribute(self.connection, attribute)
        return None

    def safely_disconnect(self):
        if self.ble is not None:
            self.ble.end_scan()
            self.ble.disconnect(0)
            self.ble.disconnect(1)
            self.ble.disconnect(2)
            self.disconnect()

    def disconnect(self):
        if self.connection is not None:
            self.ble.disconnect(self.connection)
```

6.2 Código de movimiento y comunicación de las placas

```
#configuración del movimiento en la placa Raspberry
#comunicación de la placa Raspberry y placa SSC-32

import sys
import time
import serial

sys.path.append('../lib/')

from device_listener import DeviceListener
from pose_type import PoseType

#se toma como variables de entrada las poses predefinidas (Duble_Tap, Fingers_Spread, Wave_Out, Wave_In, Fist) y se dan valores numericos (angulos) para la realizacion de movimientos
#se debe tener en cuenta que la comunicacion es tipo serial y se usa por defecto el Puerto 9600, es necesario poner antes del Puerto el '/dev/ttyS0' ya que sin eso no se puede configurar el tipo de comunicacion entre las placas

class PrintPoseListener(DeviceListener):
            def on_pose(self, pose):
        pose_type = PoseType(pose)
        print(pose_type.name)
        ssc32 = serial.Serial('/dev/ttyS0',9600, timeout=1);
        if ( pose_type.name== "FINGERS_SPREAD"):
```

```
            inicio=time.time()
            ssc32.write("#11 P500 #12 P500 #13 P500 #14 P500 #15 P500 T250 \r")
            time.sleep(2)
            ssc32.close()
            final=time.time()
            tiempo=round(final-inicio,0)
            print(tiempo)
        elif(pose_type.name== "DOUBLE_TAP"):
            inicio1=time.time()
            ssc32.write("#11 P500 #12 P2500 #13 P2500 #14 P500 #15 P500 T250 \r")
            time.sleep(2)
            ssc32.close()
            final1=time.time()
            tiempo1=round(final1-inicio1,0)
            print (tiempo1)
        elif(pose_type.name== "WAVE_IN"):
            inicio2=time.time()
            ssc32.write("#11 P2500 #12 P2500 #13 P2500 #14 P2500 #15 P500 T250 \r")
            time.sleep(2)
            ssc32.close()
            final2=time.time()
            tiempo2=round(final2-inicio2,0)
            print (tiempo2)
        elif(pose_type.name== "WAVE_OUT"):
            inicio3=time.time()
            ssc32.write("#11 P2500 #12 P2500 #13 P2500 #14 P500 #15 P2500 T250 \r")
            time.sleep(2)
            ssc32.close()
            final3=time.time()
            tiempo3=round(final3-inicio3,0)
            print (tiempo3)
        elif(pose_type.name== "FIST"):
            inicio4=time.time()
            ssc32.write("#11 P2500 #12 P2500 #13 P2500 #14 P2500 #15 P2500 T250 \r")
            time.sleep(2)
            ssc32.close()
            final4=time.time()
            tiempo4=round(final4-inicio4,0)
            print (tiempo4)
```

6.3 Código de conexión con MYO

```
#Codigo ejecutor para la coneccion con Myo y la impresión de variables de salida

import sys
import time

sys.path.append('../lib/')

from myo import Myo
from print_pose_listener import PrintPoseListener
from vibration_type import VibrationType
from pose_type import PoseType
from device_listener import DeviceListener

#ejecucion y verificacion de la comunicacion con Myo

def main():
    print('Start Myo for Linux')

    listener = PrintPoseListener()
```

```
    myo = Myo()

    try:
        myo.connect()
        myo.add_listener(listener)
        myo.vibrate(VibrationType.SHORT)
        while True:
            myo.run()

#verificacion de errores y avisos de emergencia

    except KeyboardInterrupt:
        pass
    except ValueError as ex:
        print(ex)
    finally:
        myo.safely_disconnect()
        print('Finished.')

if __name__ == '__main__':
    main()
```