ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Mecánica y Ciencias de la Producción

Implementación de la Interfaz de Selección de Algoritmos de Control para un Exoesqueleto Robótico

PROYECTO INTEGRADOR

Previo la obtención del Título de:

Ingeniero Mecánico

Presentado por:
Alejandro Miguel Unda Moreano

GUAYAQUIL - ECUADOR
Año: 2019

DEDICATORIA

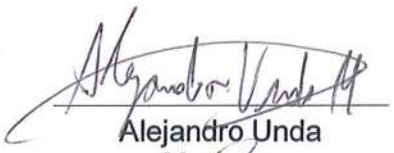
Dedico este trabajo a la memoria de mi abuela Laura Alejandrina Melendres Aguilera, quien fue en vida una segunda madre para mí y con la cual estaré eternamente agradecido por todo lo que me brindó desde mi niñez.

AGRADECIMIENTOS

Agradezco primeramente a mi familia por el apoyo que me ha otorgado y que ha sido necesario para poder llegar hasta este punto en mi carrera. Mi sincera gratitud también para con el doctor Loayza por permitirme participar de este proyecto, al club de robótica de la Espol, ROBOTA, y a Josué Mármol por su asesoría brindada.

DECLARACIÓN EXPRESA

"Los derechos de titularidad y explotación, me corresponden conforme al reglamento de propiedad intelectual de la institución; Alejandro Miguel Unda Moreano doy mi consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"



Alejandro Unda
Moreano

EVALUADORES

Ph.D. Francis Roderich Loayza

PROFESOR DE LA MATERIA

Ph.D. Francis Roderich Loayza

PROFESOR TUTOR

RESUMEN

El presente trabajo se desarrolló con el fin de implementar algoritmos de control basado en redes neuronales para mejorar la fluidez de los movimientos de un exoesqueleto de 6 grados de libertad para rehabilitación de extremidades inferiores. El dispositivo pertenece al laboratorio de Neuroimagen y Bioingeniería. Se realizó un proceso de revisión del estado inicial del software y hardware del mismo. Posterior a esta revisión se decidió cambiar el sistema de sensores analógicos de posición angular de basados en potenciómetros a sensores inerciales digitales. Se analizaron varias alternativas para este reemplazo y se determinó usar para cada articulación sensores inerciales que integraran un giroscopio y un acelerómetro de 3 ejes. Se discutieron distintas formas de integrar estos sensores en el sistema de control y se optó por una combinación de protocolos I^2C y SPI utilizando adicionalmente tres microcontroladores esclavos, situación que se decidió en base a pruebas de tiempo de muestreo. Se realizaron las adaptaciones necesarias a los algoritmos existentes para procesar las nuevas entradas de datos. Se hicieron pruebas con una pierna del exoesqueleto para verificar los resultados de las modificaciones y se compararon con resultados previos a la nueva rutina de entrenamiento, obteniendo así curvas más apegadas a los parámetros ideales del movimiento de marcha. Adicionalmente, con esta nueva implementación, se consigue que los ciclos de movimiento sean más fluidos. Se concluye que las modificaciones realizadas lograron una mejora notable en la precisión del sistema.

Palabras Clave: Exoesqueleto, Giroscopio, Acelerómetro, Red Neuronal, SPI.

ABSTRACT

The present work was developed in order to implement control algorithms that improve the fluidity of the movements of a 6-degree exoskeleton of freedom for rehabilitation of lower extremities based on neural networks found in the Bioengineering and Neuroimaging laboratory. A process of review of the initial state of the software and hardare of the same was carried out. After this review, it was decided to change the angular position sensor system from an analog one based on potentiometers to a digital one. Several alternatives for this replacement were analyzed and it was determined to use inertial sensors for each joint that integrated a gyroscope and a 3-axis accelerometer. Different ways of integrating these sensors into the control system were discussed and a combination of I2C and SPI protocols that used 3 slave microcontrollers based on sampling time tests was chosen. The necessary adaptations to the existing algorithms were made to process the new data entries. Tests were made with one leg of the exoskeleton to verify the results of the modifications and were compared with results prior to the new training routine, thus obtaining a curve more attached to the ideal parameters of the gait movement. It was concluded that the modifications made achieved a notable improvement in the accuracy of the system.

Keywords: Exoeskeleton, Gyroscope, Accelerometer, Neural Network SPI.

ÍNDICE GENERAL

ABREVIATURAS

IMU	Inertial Measurement unit
I^2C	Inter-Integrated Circuit
SPI	Serial Peripheral Interface
MPU	Magnetic Pick-Up
DOF	Degrees of Freedom
MEMS	Micro-Electromechanical Systems
PWM	Pulse-width modulation
UTP	Unshielded twisted pair
UART	Universal asynchronous receiver-transmitter

SIMBOLOGÍA

kb/s	kilobyte sobre segundo
Mb/s	Megabyte sobre segundo
Hz	Hertzio
kHz	kilohertzio

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1

1. INTRODUCCIÓN

1.1 Definición del Problema

Actualmente el Laboratorio de Neuroimagen y Bioingeniería dispone de un exoesqueleto robótico para extremidades inferiores de 6 grados de libertad. Este dispositivo se ha venido desarrollando a través de 3 proyectos de Materia Integradora, en la cual la primera parte se desarrolló el diseño mecánico y su fabricación (Begué J. & Cobeña W., 2017), en una segunda etapa se desarrolló un algoritmo de control basado en redes neuronales (Alcívar E. ,2018) y en la tercera etapa se implementó una tarjeta Raspberry pi 3 y una botonera para brindar autonomía al sistema (Mármol J. & Sánchez G., 2019). El sistema actualmente permite realizar un ciclo de movimiento, pero se descalibra y pierde la referencia al momento de realizar un segundo ciclo, por lo que es necesario corregir los algoritmos existentes. El sistema de control basado en redes neuronales permite que un algoritmo determine los parámetros necesarios para ejecutar rutinas de movimiento mediante aprendizaje basado en regresión inversa al gradiente de error. Se necesita implementar una interfaz que guarde y asocie los resultados de este entrenamiento para un determinado paciente, con el fin de que el operador pueda calibrar fácilmente el exoesqueleto a cada usuario.

1.2 Justificación del proyecto

En los últimos años el uso de soluciones robóticas para la rehabilitación de pacientes con movilidad reducida se ha extendido notablemente en la práctica médica. Para el caso de personas que han sufrido accidentes cerebrovasculares, se ha comprobado que las terapias con asistencia robótica de varios tipos ayudan a la recuperación de la movilidad de estos pacientes. (Bortole M. & Pons J.L., 2013). En concreto, la plataforma robótica en desarrollo en este proyecto ayudará a niños que tengan movilidad reducida en sus extremidades inferiores por causa de accidente o defecto congénito. Para que el exoesqueleto quede en condiciones de ser operado confiablemente por personal médico, es necesario depurar errores de continuidad en las rutinas de movimiento, habilitar una interfaz gráfica de fácil acceso y simplificar el cableado.

1.3 Objetivos

1.3.1 Objetivo General

Realizar los algoritmos de control basados en una interfaz amigable de tal forma que el exoesqueleto reproduzca los movimientos deseados.

1.3.2 Objetivos Específicos

Revisión del estado actual del dispositivo, incluyendo principalmente los algoritmos de control.

1. Reorganizar el sistema de cableado.
2. Rediseñar el algoritmo de control basado en redes neuronales (programación en Arduino y Python).
3. Elaboración de una interfaz para la selección de movimientos y algoritmos de control.

1.4 Marco teórico

1.4.1 Redes neuronales

Una red neuronal es un sistema constituido por elementos no lineales llamados nodos o neuronas cuyas entradas y salidas están interconectadas, y la magnitud o peso de cada vínculo tiene un valor que puede variar según la tarea a ejecutar. Ajustando los pesos de las conexiones entre las neuronas, el sistema en conjunto obtiene una gran versatilidad en el número de tareas específicas que puede aprender a realizar sin necesidad de una entrada directa de parámetros. Se dice que una red neuronal “aprende” cuando luego de llevar a cabo una serie de iteraciones, ha ajustado los pesos de las conexiones de forma que la salida real se acerca al comportamiento esperado dentro de la incertidumbre permisible.

Las redes neuronales responden a la necesidad de los sistemas de control de cumplir requerimientos de diseño estrictos en sistemas altamente complejos sin tener conocimiento preciso de todas las variables de la planta y el entorno. (P.J. Antsaklis,

1990). Uno de los métodos de uso más extendido por su eficacia para entrenar redes es el de propagación hacia atrás (BP por sus siglas en inglés). Este método se fundamenta en el concepto de gradiente descendente (iterar paso a paso en dirección contraria al gradiente de la función de error), definido por la regla delta o procedimiento Widrow-Hoff:

$$\forall_i \in \{1,\dots,n\}: w_i^{(new)} = w_i^{(old)} + \Delta w_i\ ; \qquad \Delta w_i\ =\ \eta(o\ -\ y)x_i \tag{1.1}$$

Donde w_i es la ponderación de la conexión entre el nodo, con salida y, y el dato de entrada x_i. Los superíndices old y new hacen referencia a las ponderaciones evaluada y nueva, respectivamente. La salida deseada es o, y η es la tasa de aprendizaje. (Kruse, 2013). El método SuperSAB se basa en comenzar con un η cercano a 1 e ir reduciéndolo conforme la salida se acerca al valor deseado.

1.4.2 Exoesqueletos

Esta palabra tiene su origen en la ciencia de la Zoología donde se usa para describir la capa externa de los artrópodos que proporciona soporte y protección a dichos animales. Un exoesqueleto mecánico es una máquina que, mediante el uso de motores, realiza trabajo que amplifica, asiste o suplanta las capacidades motrices del usuario.

Los requerimientos de diseño y potencia de un exoesqueleto dependerán directamente del tipo de actividad al que estará destinado. Los modelos dedicados a la amplificación de fuerza comúnmente son de uso industrial y requieren actuadores de mayor potencia que aquellos de uso médico. Estos últimos a su vez se encuentran en dos tipos: exoesqueletos que suplantan totalmente la movilidad perdida del usuario en sus extremidades, y exoesqueletos de uso terapéutico para rehabilitar pacientes con trastornos neuro-musculares que mantienen la movilidad en sus extremidades, pero presentan dificultad para reproducir el patrón de movimiento normal en las mismas.

Los exoesqueletos de uso terapéutico fueron el siguiente paso en el desarrollo de las órtesis activas desarrolladas entre 1935 y 1960, las cuales además de regular el movimiento del paciente mediante una armadura, proporcionaban potencia para asistir al usuario en su caminata, por medios mecánicos o hidráulicos. A finales de la década de 1960 se llevaron a cabo desarrollos en paralelo en el Instituto Mihailo Pupin en

Belgrado y en la Universidad de Wisconsin para desarrollar exoesqueletos de las extremidades inferiores de funcionamiento hidráulico. Posteriormente se optó por incorporar motores DC con el fin de mejorar la autonomía de los prototipos. (Dollar A.M. & Herr H., 2008).

En la actualidad se han patentado diversas variantes de exoesqueletos funcionales de entrenamiento para rehabilitación, tales como el LOKOMAT, LOPES, ALEX y ANDROS. El Ekso GT, primer exoesqueleto de rehabilitación para parapléjicos aprobado por la FDA, permite el ajuste individual de potencia para cada extremidad, lo cual lo hace altamente personalizable a las circunstancias de pacientes con distinto grado de afectación en sus extremidades inferiores, así como para acoplarse a aquellos que usan bastón en su caminata.

Figura 1.1. Exoesqueleto Ekso GT

Este exoesqueleto incorpora una batería que se carga en la espalda para poder realizar sus funciones de manera autónoma y dar libertad de movilidad al usuario. [https://exoskeletonreport.com/product/ekso-gt/]

Durante el desarrollo de esta tecnología, el sistema de control se ha mantenido como el principal elemento con espacio para mejoras. Los modelos comerciales más recientes han logrado reducir significativamente el tiempo de calibración necesario para adaptar el exosqueleto a un nuevo paciente. Sin embargo, el desarrollo de sus algoritmos de control no está bien documentado y no se han hecho suficientes estudios que

comparen el desempeño de distintos sistemas de control en un mismo hardware. (Young A. & Ferris D., 2017).

1.4.3 IMU

Una IMU (Inertial Measurment Unit) o Unidad de Medición Inercial es un dispositivo que integra uno o más sistemas de medición para medir la aceleración, velocidad y orientación de un sistema. Una IMU es caracterizada por sus grados de libertad, que vienen dados por la suma de los grados de libertad de cada uno de sus mecanismos. La mayoría de IMUs están compuestas por un acelerómetro y un giroscopio triaxiales, lo cual da un total de 6 grados de libertad en una IMU de este tipo.

1.4.3.1 Acelerómetro

Un acelerómetro tri-axial es un dispositivo que permite medir las componentes de aceleración de un sistema en los tres ejes espaciales. De forma particular, los acelerómetros tipo MEMS (Micro Electro-Mechanical Systems) usan un sistema de masa-resorte en cada eje para medir las aceleraciones de cada eje. El comportamiento de tal sistema está expresado por la ecuación:

$$ma = m\ddot{y} + b\dot{y} + ky \qquad \textbf{(1.2)}$$

Donde a es la aceleración inercial del sistema para un eje dado, m es la masa de la masa de prueba, y es el desplazamiento de la masa en este mismo eje, b es la constante de fricción viscosa, y k la constante del resorte.

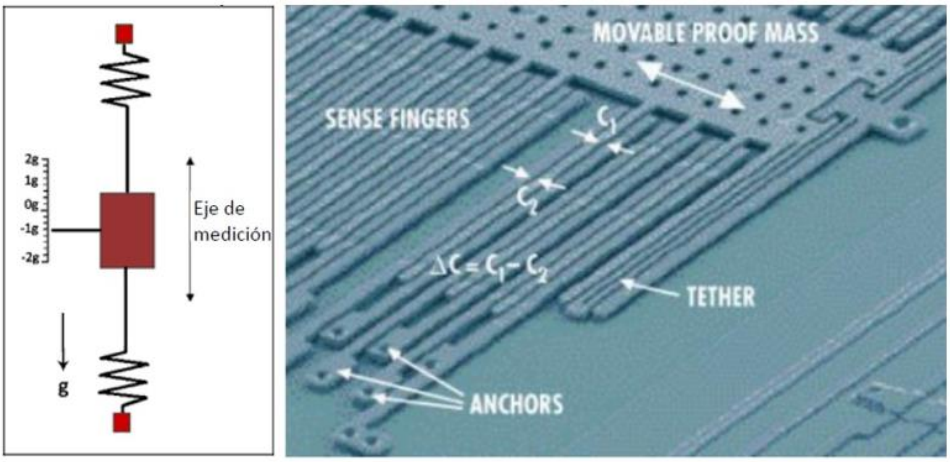


Figura 1.2. Acelerómetro MEMS integrado en un sensor

[Fuente: https://naylampmechatronics.com/]

La función de transferencia del sistema *H(s)* se expresa:

$$H(s) = \frac{y(s)}{a(s)} = \frac{m}{ms^2 + bs + k} \quad \textbf{(1.3)}$$

Donde nos interesa el funcionamiento en condiciones cuasi-estáticas (*s*=0):

$$y = \frac{ma}{k} \quad \textbf{(1.4)}$$

Y en tales condiciones, la aceleración se vuelve proporcional al desplazamiento de la masa de prueba. Como puede observarse en la figura 1.2, la masa de prueba tiene extensiones que forman parte de una combinación de capacitores de placas paralelas de distancia variable. Al moverse la masa de prueba, esto genera un cambio en la capacitancia del sistema, de acuerdo a la siguiente ecuación:

$$C = \frac{\varepsilon_0 A}{d} \quad \textbf{(1.5)}$$

Donde *C* es la capacitancia de un capacitor formado por una placa fija y un diente de la masa de prueba, $\varepsilon_0$la constante dieléctrica del vacío, *A* el área de las placas y *d* la distancia entre las mismas. De esta forma, puede transmitirse la aceleración inercial de un sistema como una señal eléctrica dependiente de esta capacitancia. Los acelerómetros pueden usarse para medir la inclinación de una superficie respecto a la tierra usando como referencia la aceleración estándar de la gravedad.

1.4.3.2 Giroscopio

Un giroscopio es un dispositivo que mide la velocidad angular de un movimiento rotatorio. Un giroscopio de 3 ejes puede usarse para determinar la orientación de un sistema con 3 grados de libertad integrando la señal de salida. Los giroscopios MEMS, al igual que los acelerómetros, funcionan con una masa de prueba que se desplaza según el movimiento del sistema, en proporción a la aceleración centrípeta causada por la rotación, dada por:

$$a_c = \omega^2 * r \qquad \textbf{(1.6)}$$

Donde a_c es la aceleración centrípeta de un sistema, ω es la velocidad angular instantánea del sistema en rotación, y *r* el radio instantáneo de rotación. La aceleración se mide con una configuración de capacitores similar a la del acelerómetro. Los giroscopios tienden a provocar un error de estado estable llamado *drift* que ocurre por la integración de la señal.

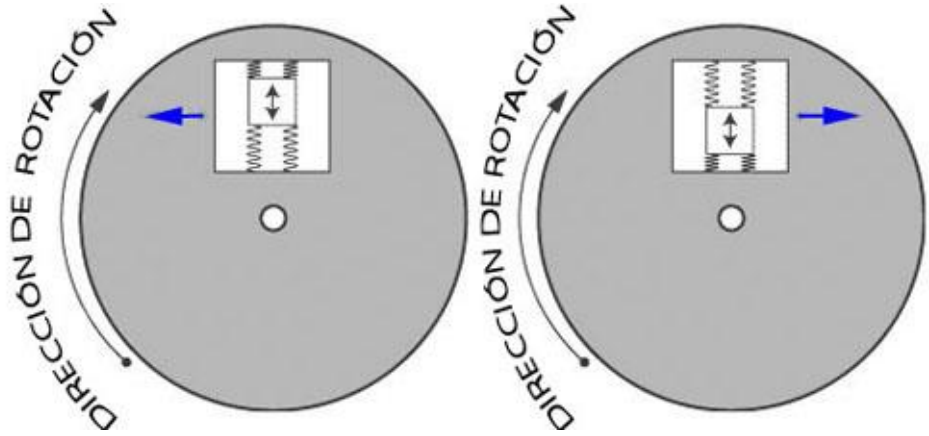


Figura 1.3. Funcionamiento de un giroscopio MEMS

[https://www.ingmecafenix.com/]

1.4.4 Protocolo I^{2}C

Este protocolo fue desarrollado en 1982 y su nombre significa “inter-integrated circuit”. Desde el año 2006 es un protocolo de uso abierto. Sus especificaciones se han modificado desde su salida para permitir mayores velocidades de transmisión. La capacidad estándar de transmisión de datos era de 100kb/s al momento de su implementación inicial, hoy en día alcanza velocidades de hasta 3.4 Mb/s en su modo de alta velocidad. Su principal uso es, como su nombre sugiere, conectar distintos

elementos dentro de la misma placa. Tiene la ventaja de solo requerir 2 alambres conectados a cada dispositivo en el bus para la transmisión de datos, además de las conexiones de alimentación y tierra. Estas dos líneas activas se denominan SCL y SDA que transmiten la señal de reloj y la señal de datos, respectivamente. Este protocolo admite cualquier cantidad de dispositivos conectados en un bus mientras difieran en su dirección de 7 bits característica. Su principal limitación es la distancia máxima del bus. Dado que se usan resistencias de pull-up en la conexión, la capacitancia de los conductores crea un filtro pasa bajos que limita la velocidad de transmisión cuanto mayor sea la longitud de los cables.

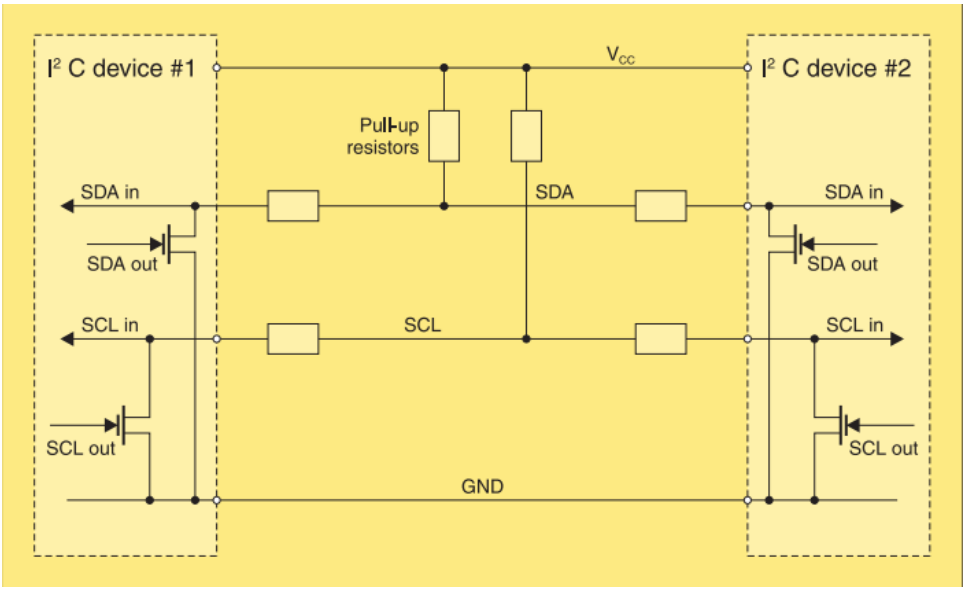


Figura 1.4. Conexión I²C uno a uno

[IEEE Instrumentation and Measurement Magazine, febrero 2009]

1.4.5 Protocolo SPI

Este protocolo fue creado en el año 1979 para el microcontrolador de 16 bits Motorola 16800. Consta de una línea de transmisión de datos del maestro al esclavo (Master-Out Slave-In), otra del esclavo al maestro (Master-In Slave-Out), una línea de señal de reloj y una línea para seleccionar el esclavo, además es necesario que los elementos compartan la misma tierra. Dado que el esclavo se selecciona mediante uno de los pines no es necesario asignar una dirección única a cada dispositivo, sin embargo, consume un pin más en el maestro por cada esclavo que se quiera agregar al bus. Cuando se desea establecer comunicación con uno de los esclavos, el dispositivo maestro selecciona el pin del mismo y lo pone en estado lógico bajo, mientras que los

pines del resto se mantienen en estado lógico alto. Este protocolo también permite la implementación de rutinas de interrupción, donde los esclavos pueden permanecer activos en otras tareas mientras no se establezca comunicación con los mismos. Admite una mayor velocidad de transmisión de datos que el I^2C, sin un límite preestablecido, pero comúnmente por encima de 10 Mb/s. (Leens F., 2009).

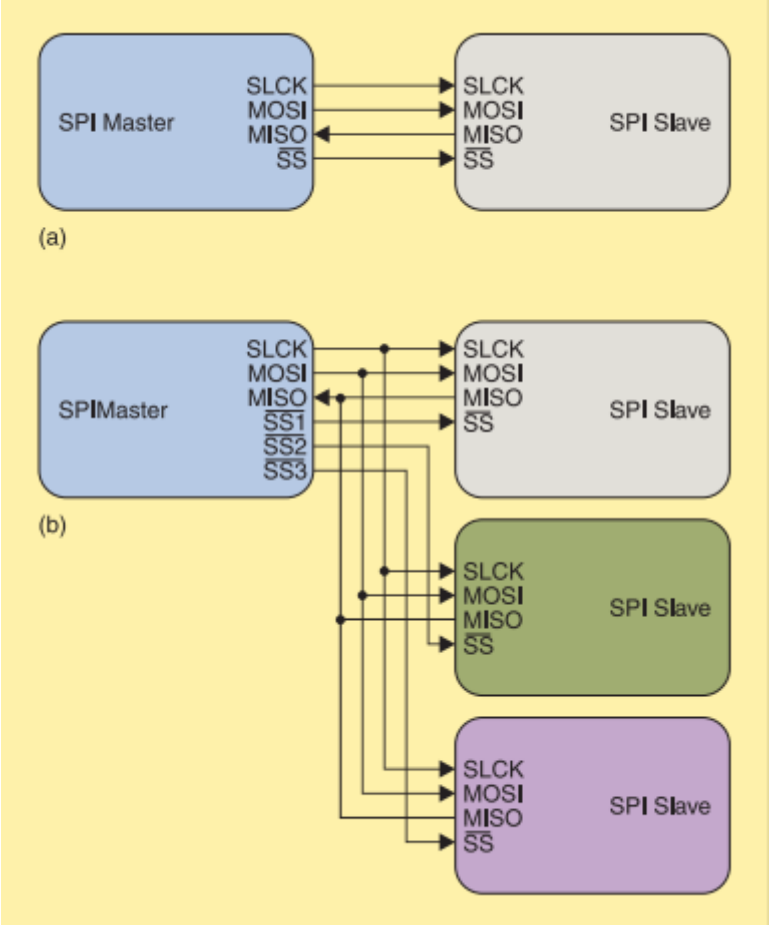


Figura 1.5. Ejemplos de conexión en SPI

[IEEE Instrumentation and Measurement Magazine, febrero 2009]

En este proyecto se ha planteado la mejora del sistema de sensores del exoesqueleto con módulos de medición inercial que incorporan tanto giroscopios como acelerómetros MEMS, y que estarán integrados al sistema de control mediante conexiones I^2C a microcontroladores intermediarios esclavos que a su vez se comunicarán en protocolo SPI con el microcontrolador maestro.

CAPÍTULO 2

2. METODOLOGÍA

Para lograr los objetivos específicos del proyecto se implementaron diversos métodos para optimizar el funcionamiento del exoesqueleto, tanto en el software como en el hardware. Se comenzó por inspeccionar el estado físico del mismo e identificar las conexiones entre sus componentes y posibles mejoras al sistema de cableado. Se revisó el algoritmo de Arduino y los drivers. Se cambió el sistema de toma de datos de posición angular y se actualizaron los algoritmos para funcionar con esta nueva entrada.

2.1 Estado inicial del exoesqueleto

Al momento de iniciar el proyecto el exoesqueleto en el laboratorio de Neuroimagen y Bioingeniería no estaba en condiciones operables pues se removieron las tarjetas Arduino Mega y Raspberry Pi 3 que originalmente estaban instaladas para dirigir el sistema de control y también estaba faltante el teclado para la operación autónoma del mismo. A continuación, se describen el resto de los componentes funcionales que estaban activos:

Figura 2.1. Estado del exoesqueleto al iniciar el proyecto

[J. Mármol, G. Sánchez, 2019]

2.1.1 Actuadores.

Para la generación de potencia el exoesqueleto cuenta con 2 motores MAXON EC 323772 (90 W, 387 Nm) en la cadera y 4 motores MAXON EC 397172 (70 W, 128Nm) para las rodillas y tobillos. La transmisión se lleva a cabo mediante un sistema de engranajes armónicos acoplados directamente al eje de cada motor, encapsulados en la armadura del exoesqueleto.

Para alimentar los actuadores se requiere una fuente DC de 24V que pueda suministrar al menos 4.39 A de corriente a cada ramal del exoesqueleto. En las tablas 2.1 y 2.2 se muestran el resto de las características proveídas por el fabricante para estos motores.

Tabla 2.1. Especificaciones del motor EC 323772

[Maxon Motor Company, 2017]

CARACTERÍSTICA	VALOR	UNIDAD
Voltaje Nominal	24	V
Velocidad Nominal	2650	RPM
Velocidad Máxima	5000	RPM
Corriente Nominal	4,39	A
Torque Nominal	387	Nm
Potencia	90	Watts
Constante de Tiempo Térmico del Devanado	52,6	s
Constante de Velocidad	135	RPM/V
Numero de par de Polos	12	--------

Tabla 2.2. Especificaciones del motor EC 397172

[Maxon Motor Company, 2017]

CARACTERÍSTICA	VALOR	UNIDAD
Voltaje Nominal	24	V
Velocidad Nominal	4860	RPM
Velocidad Máxima	10000	RPM
Corriente Nominal	3.21	A
Torque Nominal	128	mNm
Potencia	70	Watts
Constante de Tiempo Térmico del Devanado	29.6	s
Constante de Velocidad	259	RPM/V
Numero de par de Polos	8	--------

2.1.2 Controlador.

Los 6 motores del exoesqueleto son controlados por módulos ESCON36/3 EC, las cuales son las tarjetas recomendadas por el fabricante de los motores. En el laboratorio se encuentra una PC con el programa ESCON Studio para programar dichos módulos para que puedan ser usados como reguladores de la velocidad y torque del motor. El controlador traduce las señales PWM (pulse-width modulation) en valores de velocidad angular que desarrollará el motor. Estos controladores mantienen la velocidad que se les indica mediante un lazo de control cerrado interno. También pueden funcionar en lazo abierto o por control de corriente.

Tabla 2.3. Especificaciones del driver ESCON 36/3 EC

[Maxon Motor Company, 2017]

CARACTERÍSTICA	VALOR	UNIDAD
Voltaje Nominal de Operación	10…36	V
Corriente de salida (I_{cont}/I_{max})	2.7/9	A/A
Voltaje de Alimentación a Sensores Hall	5	V
Frecuencia de modulación por ancho de pulso.	53.6	kHz
Frecuencia de muestreo para el controlador PI de corriente.	53.6	kHz
Frecuencia de muestreo para el controlador PI de corriente.	5.36	kHz
Temperatura de Operación	-30…+45	°C

2.1.3 Sensores.

Acoplado al eje de salida de cada motor se encontraban instaladas unas poleas dentadas que transmiten la velocidad de giro a un potenciómetro mediante una banda sincrónica. Estos potenciómetros se usan para medir el desplazamiento angular del eje del motor pues su resistencia es proporcional a la posición angular del mando conectado mecánicamente al eje. Se genera entonces una señal analógica de entre 0 y 5V que es luego leída por el Arduino para convertirse en una entrada que utilizará el algoritmo de red neuronal para ejecutar la rutina.

Figura 2.2. Potenciómetro acoplado a la cadera izquierda

[J. Mármol, G. Sánchez, 2019]

2.2 Cambio de sensores de posición angular

Luego de revisiones en el sistema mecánico y mediciones a través del sistema de adquisición de datos, se pudo observar que el sistema de sensores analógicos era una fuente muy alta de entrada de ruido en el sistema de control. Adicionalmente, las bandas sincrónicas presentaban distintos grados de desajuste y holgura lo cual hace que no provean una lectura adecuada a tiempo real del desplazamiento angular de los motores. Es por esto que se tomó como objetivo específico el reemplazo del sistema de sensores. A continuación, se describen las alternativas propuestas para este fin.

2.2.1 Encoder óptico absoluto

En un encoder óptico, una fuente emite luz directamente o a través de un disco óptico rallado para que la luz pase a su través o quede bloqueada. Un detector óptico, o cabeza de lectura, detecta el paso de la luz y genera la señal eléctrica correspondiente. Rejillas ópticas están dispuestas como una serie de marcas que pueden utilizarse para medir un ángulo o un movimiento. La escala de las marcas puede ser muy fina – hasta de micrones – lo que permite que muchos encoders ópticos puedan medir con altos niveles de precisión. Sin embargo, tienen un precio relativamente alto, partiendo desde los 30 dólares.

2.2.2 Encoder magnético

Este tipo de encoder usa una serie de polos magnéticos para representar la posición del encoder a un sensor magnético (típicamente magneto-resistivo o de efecto Hall). Este sensor lee las posiciones del polo magnético. A diferencia de los encoders ópticos, los sensores Hall al ser magnéticos pueden trabajar en entornos extremos (temperatura, salpicadura de fluidos, grasas, etc.) y polvorientos; no hará falta mantenimiento de limpieza. Su precio es significativamente menor que el de los encoder ópticos, habiendo gamas desde 8 dólares la unidad (no incluido el costo del imán).

2.2.3 Módulo MPU 6050

EL MPU6050 es una unidad de medición inercial o IMU de 6 grados de libertad pues combina un acelerómetro de 3 ejes y un giroscopio de 3 ejes. Este sensor es muy utilizado en navegación, goniometría, estabilización, etc. Este módulo está

específicamente diseñado para trabajar con tarjetas Arduino y provee mediciones analógicas precisas de velocidad y aceleración angular. Al acoplarse a una superficie es posible usarlo para medir la inclinación de la misma con respecto a la tierra.

A continuación, se muestra la matriz de decisión que se utilizó para para ponderar las ventajas de cada alternativa.

Tabla 2.4. Matriz de decisión para la implementación de sensores

[Elaboración propia]

Variables	Peso (%)	Sensor		
		Encoder óptico absoluto	Encoder Magnético absoluto	MPU 6050
Fiabilidad	40	1.00	1.00	0.80
Facilidad de uso e instalación	30	0.90	0.60	0.90
Costo	30	0.30	0.80	1.
Total	100	0.76	0.82	0.89

De acuerdo al total ponderado, el módulo MPU 6050 resulta por ende el más conveniente para el proyecto por proveer una fiabilidad suficiente a bajo costo y con una mayor facilidad de instalación que el encoder magnético.

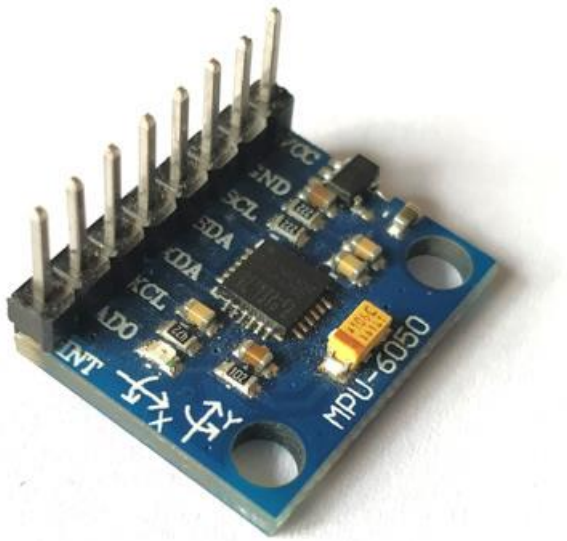


Figura 2.3. Sensor MPU 6050

Este sensor funciona con protocolo I^2C y contiene un giroscopio y un acelerómetro de 3 ejes. Tiene una frecuencia de muestreo máxima de 1kHz. [https://components101.com/sensors/mpu6050-module]

2.3 Pruebas con el sistema MPU

El acelerómetro del MPU 6050 permite determinar la inclinación vertical con bastante exactitud en reposo mediante el cálculo de las componentes de la gravedad en sus ejes. Sin embargo, al someterse a aceleraciones externas, lo cual es inevitable en el sistema a tratar, genera un ruido muy alto, pues como se mencionó en el capítulo 1, su función de transferencia se estima para bajas frecuencias. Para calcular el ángulo de inclinación con el acelerómetro se usa la siguiente relación geométrica:

$$\theta_{ac} = atan\ \frac{(a_y)}{\sqrt{a_x^2+a_z^2}} \qquad \textbf{(2.1)}$$

Donde θ_{ac} es el ángulo de inclinación del eje x respecto a la vertical, y a_x, a_y y a_z son las aceleraciones medidas en los tres ejes respectivos.

El giroscopio puede obtener la posición angular integrando las mediciones de velocidad angular, pero este método conlleva un error conocido como drift.

2.3.1 Filtro complementario

Para aplacar el efecto de los errores antes mencionados, se decidió implementar un filtro complementario que determina el valor de la posición angular como un ponderado de las mediciones de ambos instrumentos. Este filtro toma como entrada los valores de velocidad angular en el eje X del giroscopio, y aceleración angular del eje Y y Z para el acelerómetro, ya que sólo se requiere la posición angular en el plano vertical. Con estos datos se procedió a calcular la posición angular de dos formas independientes, integrando la señal de velocidad, y obteniendo las componentes relativas de la aceleración de la gravedad. La ecuación que determina el filtro es la siguiente:

$$\theta_x = 0.98 * (\theta_{old} + \omega_x * \Delta t) + 0.02 * \theta_{ac} \quad \textbf{(2.2)}$$

Donde θ_x es el ángulo de inclinación respecto a la vertical, θ_{old} el ángulo calculado durante el intervalo de medición anterior, ω_x la velocidad angular obtenida por el giroscopio en el eje x, Δt la duración del intervalo y θ_{ac} el ángulo calculado por el acelerómetro. La duración del intervalo se obtiene con la función *millis* del Arduino.
Para calibrar el acelerómetro se usa una de las funciones propias de la librería Wire para corregir el offset del sensor. Se deja el sensor en reposo apoyado contra una superficie lisa; y se establecen los offsets del acelerómetro a cero, menos el eje X que se establece a 16384, el equivalente a la aceleración de la gravedad en la escala usada por el sensor.

2.3.2 Conexiones

Considerando que el MPU-6050 funciona con el protocolo de comunicación I^2C, este solo puede funcionar dentro de un BUS donde cada dispositivo tenga una dirección I^2C distinta. Sin embargo, por su condición de fábrica este módulo solo puede ser fijado a dos direcciones distintas, 0x68 y 0x69 (Hexadecimal). Esto representó un reto para el proyecto puesto que se requerían 6 sensores en las articulaciones del exoesqueleto que midan la posición angular simultáneamente en los eslabones y alimenten con estos datos al algoritmo de red neuronal en tiempo real.

Se realizaron pruebas con un multiplexor 74HC157 que realizara un barrido de las conexiones. Esto es, que la tarjeta Arduino Mega solo se comunica con un MPU-6050 a

la vez, alternando el sensor que se encuentra en la dirección activa (0x68) mientras el resto se coloca en la dirección inactiva (0x69) al fijar su terminal AD0 a nivel alto. Sin embargo, esto creaba un retardo importante en la captación de datos puesto que cada vez que el Arduino alternaba entre sensores había que sincronizarse con el reloj interno de cada uno de estos. Además, la tarjeta Arduino habría tenido que procesar el algoritmo de filtro complementario para cada uno de los 6 sensores al mismo tiempo que maneja la comunicación con los drivers y la Raspberry Pi. Por lo tanto, la frecuencia de muestreo sería más lenta aún al implementar esta rutina de medición durante el funcionamiento del exoesqueleto.

Dado este inconveniente, se consideró usar un multiplexor TCA9548A especialmente dedicado a dispositivos I^2C con la capacidad de enmascarar las direcciones de dispositivos iguales, sin embargo, se abandonó esta alternativa porque dicho elemento tenía que importarse y no llegaría en el tiempo previsto.

La siguiente alternativa fue usar tarjetas Arduino distintas para cada sensor y que estos alimentaran los datos a la Raspberry Pi por medio de su puerto serial. Este método, aunque requería de más microcontroladores y cableado, proporciona la forma más directa de tratar con varios dispositivos I^2C a la vez. No obstante, se observó que se podía optimizar el algoritmo haciendo que la comunicación fuera entre dispositivos Arduino antes que entre Arduino y Raspberry. Por esta razón, la disposición que se decidió definitivamente fue que 3 tarjetas Arduino UNO captarán datos de 2 sensores cada uno mediante protocolo I^2C, y que estos datos fueran transferidos a la tarjeta Arduino Mega usando el protocolo SPI, que admite comunicación de varios esclavos con un maestro. Con esto además se distribuyó la carga de procesamiento entre varios microcontroladores y el Arduino Mega solo tendría que manejar los datos entrantes, lo que requiere menos cambios en el algoritmo principal comparando a que se realice la comunicación con la tarjeta Raspberry.

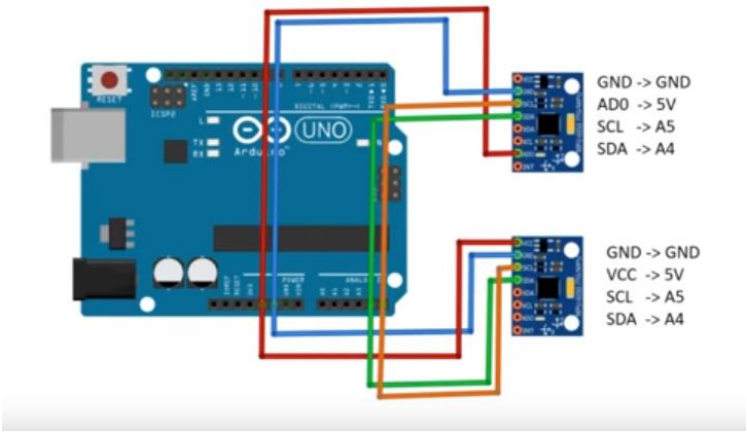


Figura 2.4. Configuración del Arduino UNO con 2 MPU 6050.

En la configuración mostrada los MPU-6050 se conectan en paralelo a los pines GND, 5V, A4 y A5 del Arduino UNO, y se fija uno de los pines AD0 del MPU a HIGH para que cambie su dirección de 0x68 a 0x69. [Elaboración propia]

2.4 Modificación a la estructura de hardware y software del sistema de control.

El sistema de control del exoesqueleto está basado en un algoritmo de redes neuronales que se ejecuta a través de una PC o tarjeta Raspberry, el cual procesa información compilada por la tarjeta Arduino Mega, la cual recoge señales de las tarjetas Arduino UNO que colectan información de los sensores de posición angular. A su vez la tarjeta Arduino Mega envía una salida a los controladores para que aumenten o disminuyan la velocidad angular de cada motor. En la figura 2.5 se detalla el diagrama de flujo de información para el sistema de control del exoesqueleto completo.

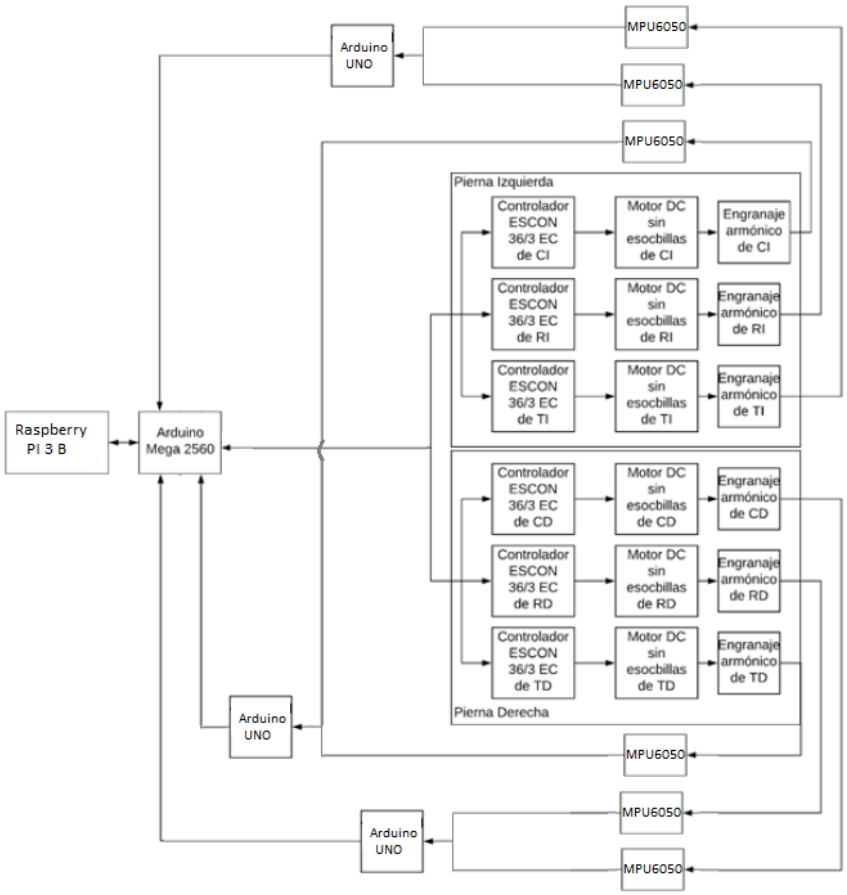


Figura 2.5. Diagrama de conexión y jerarquía del software y hardware

[Elaboración propia]

Tabla 2.5 Conexiones en la tarjeta Arduino Mega

[Elaboración propia]

No. Pin	Función	Descripción
28	Pin de habilitación para tobillo.	Habilita el envío de señal PWM desde el controlador al motor del tobillo derecho
29	Pin de dirección para tobillo.	Establece el sentido de giro del motor en el tobillo derecho
10	Pin de envío de señal PWM para tobillo.	Envía la señal PWM al controlador conectado al motor en el tobillo derecho.
30	Pin de habilitación para rodilla.	Habilita el envío de señal PWM desde el controlador al motor de la rodilla derecha
31	Pin de dirección para rodilla.	Establece el sentido de giro del motor en la rodilla derecha.

9	Pin de envío de señal PWM para rodilla.	Envía la señal PWM al controlador conectado al motor en la rodilla derecha.
32	Pin de habilitación para rodilla.	Habilita el envío de señal PWM desde el controlador al motor de la cadera derecha
33	Pin de dirección para rodilla.	Establece el sentido de giro del motor en la cadera derecha.
11	Pin de envío de señal PWM para rodilla.	Envía la señal PWM al controlador conectado al motor en la cadera derecha.
51	Pin SPI MOSI Master Out Slave In	Envía instrucciones a los Arduinos esclavos
50	Pin SPI MISO Master In Slave Out	Recibe datos de los Arduinos esclavos
52	Pin SPI SCK	Envía la señal de reloj a los Arduinos esclavos
53	Pin Slave Select 1	Pin para habilitación de esclavo del Arduino de la cadera derecha.
49	Pin Slave Select 2	Pin para habilitación de esclavo del Arduino de la rodilla y tobillo derechos.

Acorde a lo discutido en la sección anterior, se decidió reemplazar los potenciómetros por módulos MPU 6050 para la medición de posición angular. Esto por ende requirió cambios en la entrada del algoritmo de red neuronal además de la implementación de un nuevo algoritmo de recolección de datos.

2.5 Determinación de los parámetros de trabajo en el sistema de red neuronal

Igual que en proyectos anteriores, se han usado los datos de Bovi, Rabuffetti, Mazzoleni y Ferrarin (2011) como referencia al movimiento patrón de caminata en jóvenes. En la figura 2.6 se detallan las gráficas ángulo de flexión versus porcentaje de ciclo de caminata para las articulaciones de las extremidades inferiores pertinentes al proyecto. A partir de estos datos se generó una matriz de referencia de velocidades objetivo que son las salidas deseadas por cada uno de los elementos en la red neuronal.

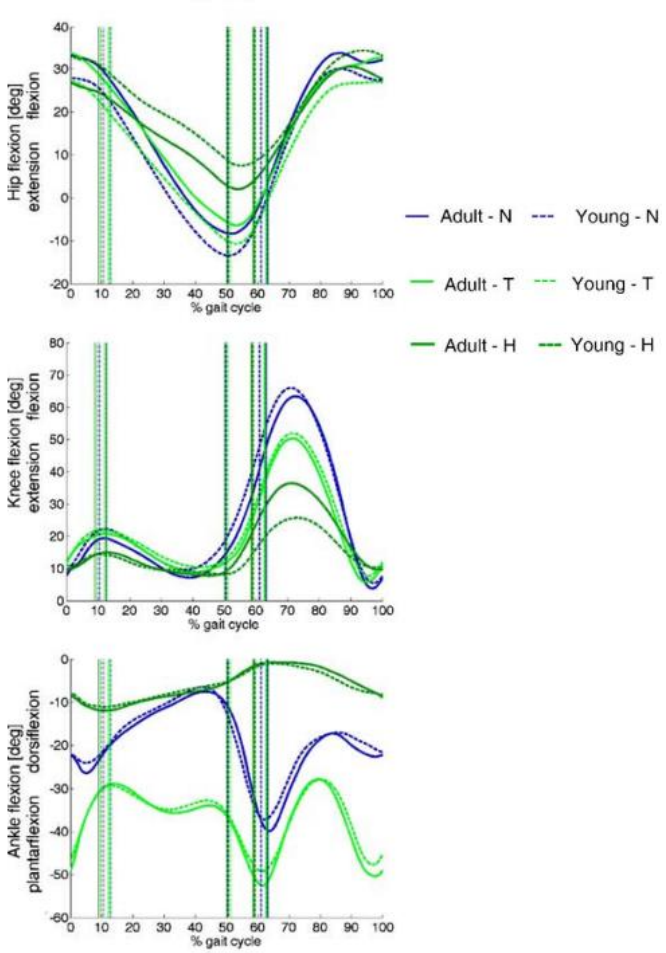


Figura 2.6. Cinemática del ciclo de caminata.

Se muestran ciclos para movimiento normal (azul), sobre las puntas de los pies (verde claro) y sobre los talones (verde oscuro). Las líneas verticales indican los instantes donde el pie despega del suelo (izquierda) y cuando vuelve a tocarlo (derecha). [Bovi et al., 2011.]

Para empezar el ciclo de entrenamiento primero es necesario hacer una rutina de prueba que tome una muestra el tiempo que toma a los motores ir desde una posición cero a un ángulo de 90° a velocidad máxima (255). Esta subrutina generará un vector de factores F que servirán para la fase de entrenamiento como tal.

Luego de haber calculado estos valores, se procede a una fase de entrenamiento en la cual cada neurona es asignada a un fragmento de ciclo y, mediante el método de propagación hacia atrás súper SAB se obtendrán los valores w para cada neurona del proceso, empezando desde una estimación de valores iniciales generados al azar. El error instantáneo en cada neurona, es decir la diferencia de velocidades angulares respecto de la requerida para cada fragmento de trayectoria se calcula de la siguiente forma:

$$\varepsilon_i = (o_i - y_i) = (\theta_{i+1} + \theta_i)/t_{del} - y_i \quad \textbf{(2.3)}$$

Donde θ_i y θ_{i+1} se obtienen de una hoja de cálculo con los valores presentados en la figura 2.6, modificados para medirse con respecto a la vertical, y t_{del}es una centésima del tiempo fijado por el usuario para la duración del ciclo de caminata. El error es la diferencia absoluta del valor actual de velocidad angular y_i respecto de o_i.

Para este método en particular, se tiene que el valor η se actualiza en cada iteración según el siguiente criterio: se multiplica por 0.6 si el producto del error actual por el error evaluado en el paso anterior es negativo, y se multiplica por 1.07 si este producto es positivo. Esto significa que si el error cambió de signo es porque el algoritmo está convergiendo a la solución y por tanto debe bajar su η para aumentar la precisión, si el error no ha cambiado de signo es porque todavía está lejos de converger y por tanto es recomendable aumentar la tasa de aprendizaje para dar saltos más grandes en cada iteración.

$$\eta^i = 0.6\,\eta^{i-1}; \quad \varepsilon^i * \varepsilon^{i-1} < 0 \quad \textbf{(2.4)}$$

$$\eta^i = 1.07\,\eta^{i-1}; \quad \varepsilon^i * \varepsilon^{i-1} > 0 \quad \textbf{(2.5)}$$

Con estos valores de eta se procede a calcular los valores w de acuerdo a lo enunciado en la ecuación (1) en el capítulo 1.

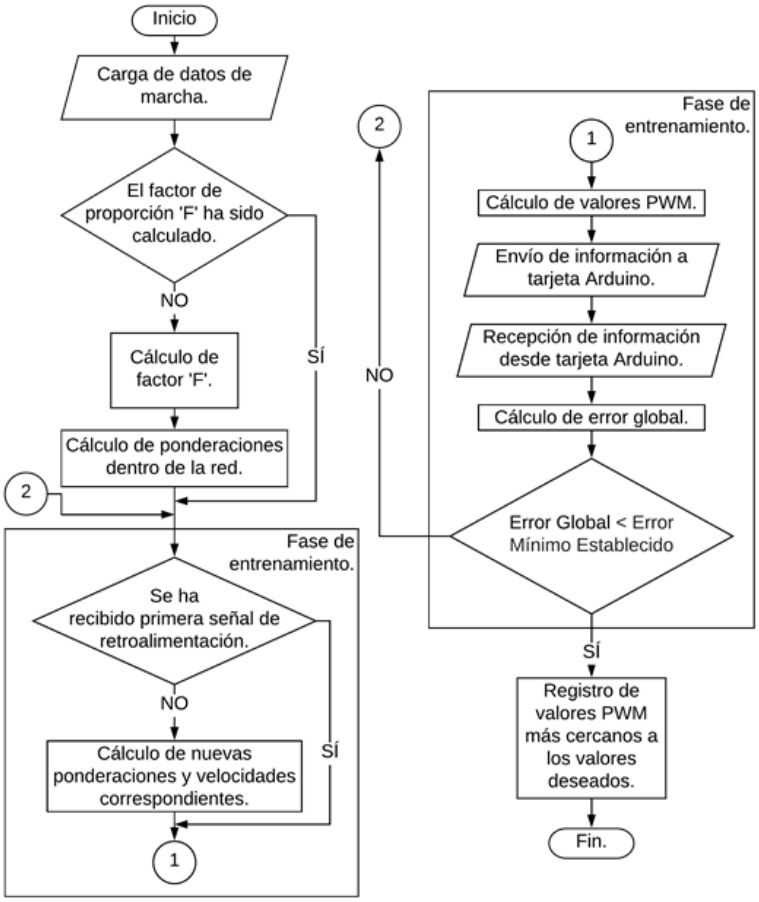


Figura 2.7. Diagrama de flujo de la rutina de entrenamiento

[E. Alcivar, 2018]

2.6 Evaluación del desempeño de la rutina

Una vez ejecutada la rutina, los datos de retroalimentación fueron recogidos por la tarjeta Raspberri Pi y se pasaron a una PC con el programa Origin para su análisis. Para determinar qué tanto se acercan los datos reales de movimiento a los valores deseados, se usó el coeficiente de correlación de Pearson r_{xy}, que se define de la siguiente manera para dos conjuntos de datos X y Y de n elementos cada uno:

$$r_{XY} = \frac{\sum x_i\, y_i - n\bar{x}\bar{y}}{(n-1)s_x s_y} \qquad \textbf{(2.6)}$$

Donde $\bar{x}$ y $\bar{y}$ son las medias de los conjuntos de datos a comparar, y s_x y s_y son sus desviaciones estándar respectivas. Este coeficiente arroja valores entre -1 y 1, y mientras más cercano a 1 sea, mayor es la correlación entre X y Y.

2.7 Consideración de Control tradicional PID

Se consideró revertir el funcionamiento del sistema de control a uno definido por un algoritmo PID. Sin embargo, se descartó esta alternativa por su baja eficiencia computacional para la tarea requerida. Un controlador tipo PID es aquel que tiene un componente de control proporcional al error, uno dependiente de la integral del error, y uno dependiente de la derivada de la función error. Su ecuación característica y función de transferencia son las siguientes:

$$u(t) = K_p e(t) + \frac{K_p}{T_i}\int_0^t e(\tau)d\tau + K_p T_d \frac{de(t)}{dt} \quad \textbf{(2.7)}$$

$$C_{PID}(s) = K_p(1 + \frac{1}{T_r s} + T_d s) \quad \textbf{(2.8)}$$

Donde e(t) es la función de error, K_p es la constante proporcional, T_r el tiempo de reset, y T_d el tiempo derivativo. Las propiedades de este método lo hacen ideal para hacer que la salida de un sistema converja rápidamente hacia un punto de operación el cual es fijo, y se mantenga cercano a ese valor. En este caso se tienen varias variables de salida que son los ángulos en cada una de las articulaciones, y la referencia es móvil puesto que se espera que el desplazamiento angular sea distinto en cada instante, según los datos recolectados sobre cinemática del movimiento de caminata. Para hacer que un algoritmo PID se ajuste a la trayectoria requerida, deberían calcularse las 3 constantes K_p, T_r y T_d para cada uno de los fragmentos de ciclo de caminata, y debería lograrse para cada intervalo un tiempo de estabilización más corto que el intervalo de tiempo entre una posición y la siguiente. Es relevante mencionar que los drivers de los motores cuentan con su propio lazo de control cerrado PID para mantener una velocidad constante cuando esta es requerida.

2.8 Rediseño del cableado

Se determinó que los MPU6050 se coloquen en las caras externas de ambos lados de la armadura, para estar ubicados junto los controladores de los motores y permitir un acceso fácil a ambos además de unificar el cableado. Ya que tanto los drivers como los MPU requieren 4 cables de conexión, se consideró que lo ideal sería usar cables de par trenzado UTP, ya que una línea contiene 8 pares. De esta forma se requeriría solo una línea UTP para cada articulación que contenga el cableado de los drivers y los

sensores. Los MPU se montaron en placas de baquelita a las que están unidas por soldadura blanda, por lo cual están firmemente montados a la estructura, pero hace fácil desmontarlos cuando se necesite.

Dado que el proyecto requirió de más microcontroladores, se usaron más cables USB-serial para alimentar energía a las tarjetas Arduino UNO. Además, se requirió de un Hub USB para que la tarjeta Raspberry pudiera abastecer a todos los Arduino además de estar conectada a un teclado y mouse, pues esta solo tiene 4 puertos USB.

CAPÍTULO 3

3. RESULTADOS Y ANÁLISIS

En la siguiente sección se presentan los resultados de las pruebas descritas en la sección de metodología para la recolección de datos, así como resultados para el movimiento de marcha a paso natural. El motor de la cadera de la pierna izquierda quedó deshabilitado por una rotura en el acople. Por este motivo se trabajó solo con resultados de la pierna derecha.

3.1 Pruebas de frecuencia de muestreo para la selección de sensores

Se crearon rutinas de prueba para la toma de datos de los MPU en configuraciones de tres sensores. Haciendo la toma de datos con multiplexor, de Arduino a Raspberry mediante puerto Serial-UART, y de Arduino a Arduino mediante SPI. En la tabla a continuación se describe el desempeño de las distintas configuraciones de acuerdo al tiempo de muestreo medido por la función millis de la tarjeta Arduino:

Tabla 3.1. Frecuencias de adquisición de datos para distintas conexiones.

[Elaboración propia]

Configuración usada.	**Frecuencia promedio de un ciclo completo de actualización de datos para distinto número de sensores activos.**		
	1 sensor	**2 sensores**	**3 sensores**
Arduino único con multiplexor on/off	120 Hz	45 Hz	18 Hz
Arduinos UNO conectados a Raspberry PI (serial)	300 Hz	300Hz	140 Hz
Arduinos UNO conectados via SPI a Arduino Mega	200 Hz	200 Hz	90 Hz

En la tabla se puede apreciar que el modo de conexión de Arduino UNO a Raspberry produce la frecuencia de muestreo más alta de los tres métodos, ya que la comunicación por puerto serial-UART es más rápida que la SPI. Sin embargo, esto no toma en cuenta que luego de recolectar los datos, estos deben ir eventualmente a la

tarjeta Arduino Mega para ser usados en los cálculos de la rutina de entrenamiento y calibración, lo cual supondría otro paso más de transmisión con un consecuente retraso comparado a que si los datos fueran recolectados directamente en la tarjeta Arduino Mega. Además, el uso del puerto serial causa interrupciones en el procesamiento de esta última mientras ejecuta su rutina de control principal. Otro aspecto a notar es que, a diferencia del método de barrido, no se presenta diferencia en la frecuencia de muestreo para uno y dos sensores en las otras configuraciones. Esto es porque las rutinas se ejecutan igual para dos sensores conectados a una misma tarjeta Arduino UNO. La diferencia solo se nota cuando se añade más de un Arduino en la conexión. En el método de barrido, sin embargo, cada sensor que se añade al sistema representa una pérdida en la frecuencia total de muestreo puesto que la conexión siempre es de uno a uno. Puesto que se desea una frecuencia de muestreo de al menos 10 Hz para el funcionamiento adecuado del sistema, se consideró inadecuado el método de multiplexor de switch ya que 6 MPUs hubieran dado como resultado una frecuencia aún menor a esta.

Un resultado que cabe mencionar de estas pruebas es que luego de un período de entre 2 y 3 minutos, la lectura de los MPU mostrada en el serial del maestro se quedaba congelada, aunque se movieran las articulaciones. En estos casos se restablecía la lectura al presionar el botón de reinicio de las tarjetas Arduino UNO. Es probable que la causa fuese un desbordamiento de datos en las tarjetas recolectoras, sin embargo, no se pudo determinar con certeza la causa de este fenómeno ni evitar su ocurrencia, aunque se añadieran comandos para limpiar el buffer. Luego de dejar los algoritmos corriendo durante unos minutos, eventualmente las lecturas se congelaban siempre y se tenía que resetear las tarjetas Arduino. Es probable también que la rutina de interrupción que usan los controladores esclavos para comunicarse con el Arduino Mega impidiese que retomen su rutina normal de recolección de datos de los MPU por algún desfase en el reloj, ya que este problema no se presenta leyendo los datos directamente de las tarjetas esclavas por medio del puerto serial.

3.2 Pruebas de ciclo de caminata

Después de haber terminado las pruebas con el sistema de sensores y determinar la configuración más idónea, se procedió a realizar las pruebas con el algoritmo de red

neuronal modificado para la compatibilidad con los giroscopios. Se realizaron cambios en la base de datos de los patrones de caminata pues estas estaban dadas en ángulos relativos de flexión y extensión. Se cambió esto para que los valores sean comparados a ángulos absolutos medidos desde la vertical, como se muestra en la tabla 3.2. Para el ángulo absoluto de tibia se restó el ángulo de flexión de la cadera menos el ángulo de flexión de la rodilla, mientras que para el ángulo plantar se sumó el ángulo absoluto de la tibia al relativo del tobillo, y se sumarron 20 grados que representan la anchura del pie entre la planta y el empeine. También se modificó el algoritmo de obtención de factores F para que no se tenga que realizar un recorrido muy largo para determinar la relación entre señal PWM y velocidad efectiva.

$$Ángulo\ tibia\ absoluto = Ángulo\ cadera - Ángulo\ flexión\ rodilla \quad \textbf{(3.1)}$$

$$Ángulo\ pie\ absoluto = Ángulo\ tibia + Ángulo\ flexión\ tobillo + 20° \quad \textbf{(3.2)}$$

Tabla 3.2 Salidas de ángulo deseadas

[Elaboración propia]

% ciclo de caminata	ángulo de flexión cadera (°)	ángulo de flexión rodilla (°)	ángulo tibia absoluto (°)	ángulo de flexión tobillo (°)	ángulo plantar absoluto (°)
0	27.9	8.8	19.1	-21.9	17.2
10	24.8	21.7	3.1	-21.8	1.3
20	13.9	18.6	-4.7	-14.2	1.1
30	2.1	12.2	-10.1	-10.5	-0.6
40	-7.9	9.9	-17.8	-6.8	-4.6
50	-13.4	18.4	-31.8	-12.5	-24.3
60	-5.5	44.4	-49.9	-35.9	-65.8
70	14.0	65.8	-51.8	-27.5	-59.3
80	27.6	53.3	-25.7	-18.2	-23.9
90	29.5	19.2	10.2	-18.0	12.2
100	27.3	7.5	19.8	-21.7	18.1

En la figura 3.1 se observan los resultados del movimiento para una secuencia de entrenamiento con valores w del proyecto anterior (en rojo) y para la rutina corregida con realimentación de posición angular (en azul). Los gráficos se han tomado con un solo ciclo por facilidad de comparación con los datos anteriores. En la figura puede notarse que la rutina con los valores del entrenamiento viejo presenta picos, lo cual puede deberse a incorrecta asignación de valores para la dirección de giro del motor lo que hace que cabecee de un lado a otro cuando tiene que mantener su posición. Estos errores se notan principalmente en los picos de la función de movimiento, y en la

meseta cercana a cero para el movimiento del tobillo. En comparación, las curvas que representan la salida con la nueva rutina de entrenamiento son más suaves y convergen satisfactoriamente a los valores esperados de las curvas teóricas.

Se realizó un análisis de correlación de Pearson a las 3 articulaciones activas comparando los datos del entrenamiento anterior y el nuevo con los valores angulares esperados en los fragmentos de ciclo. Usando la ecuación (11) enunciada en el capítulo 2, se hallaron las correlaciones entre las distintas rutinas. Se tiene que para la cadera, la media de los valores deseados es 11.42, con una desviación estándar de 15.57; mientras que la distribución de valores reales medidos del primer entrenamiento tiene una media de 11.46 y una desviación estándar de 15.62. Reemplazando estos valores en la ecuación (2.6):

$$r_{XY} = \frac{\sum x_i\, y_i - 100 * (11.42) * (11.46)}{(99) * (15.57) * (15.62)}$$

$$r_{XY} = \frac{34410.75 - 13087.32}{24077.14}$$

$$r_{XY} = 0.88563$$

Donde $\sum x_i\, y_i$ es la sumatoria de los productos de los valores correspondientes en cada conjunto de datos. De esta forma se obtuvieron las correlaciones para cada articulación. Los resultados de este análisis se muestran en la tabla 3.3:

Tabla 3.3 Coeficientes de Pearson de correlación a los valores angulares esperados

[Elaboración propia]

	Cadera	**Rodilla**	**Tobillo**
Anterior	0.88563	0.99664	0.98734
Nuevo	0.97879	0.99987	0.99979

La mejora más notoria se dio en la cadera, la cual tenía un coeficiente de Pearson bastante alejado del 1. En las otras articulaciones sigue habiendo una relativa mejora del coeficiente, aunque estos ya estaban bastante cercanos a la unidad

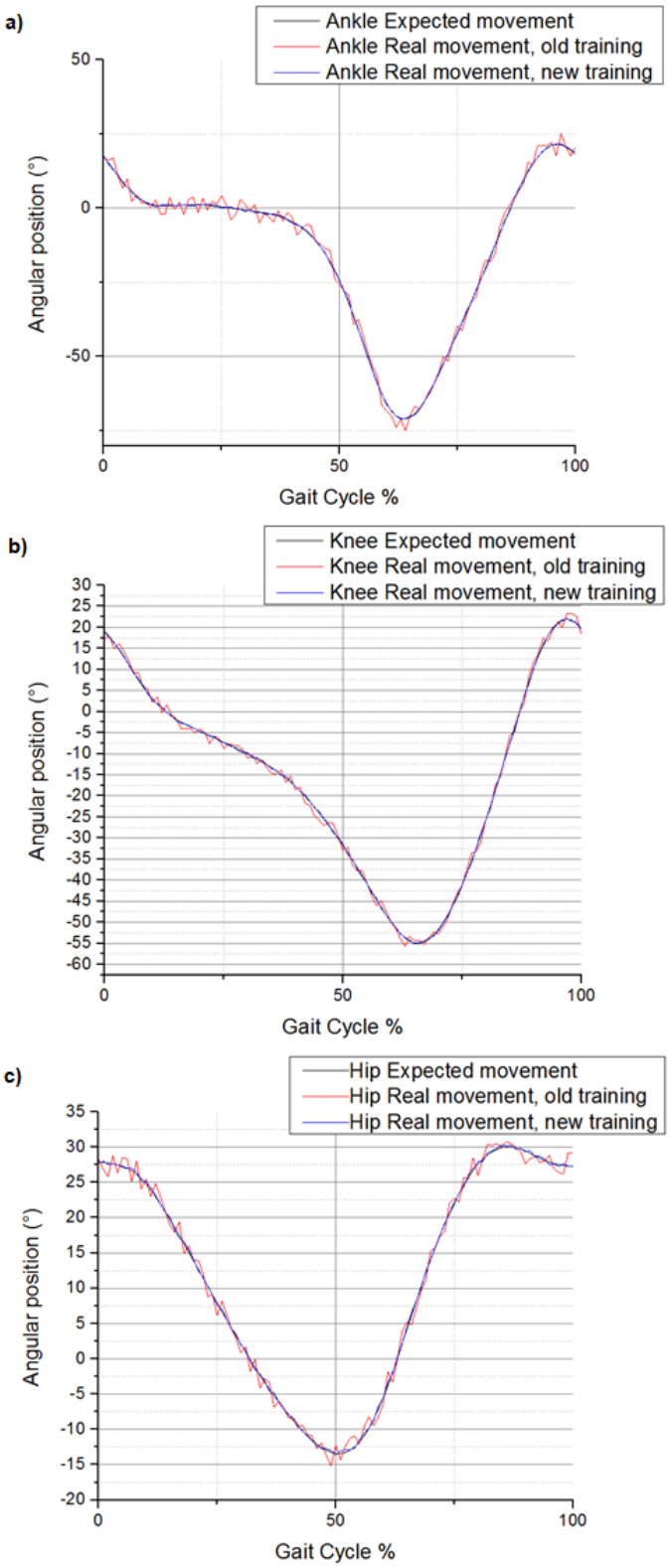


Figura 3.1. Resultados para rutina de marcha

Posición angular versus porcentaje de ciclo para las distintas articulaciones: a) tobillo, b) rodilla y c) cadera. [Elaboración propia]

Como se mencionó antes, el sistema de sensores debía reiniciarse luego de un lapso de 2 a 3 minutos de actividad continua mediante el botón de reseteo de los Arduino UNO. En caso de no hacerlo, el sistema llegaba a una posición en la cual cree que debe seguir moviendo uno de los motores hacia una dirección, aunque ya haya agotado su rango de movimiento ya que le llega una lectura desactualizada. En esos casos se debió cortar la energía para prevenir daños al motor. También se intentó realizar una rutina de sentado, sin embargo, esto no fue posible ya que el motor de la cadera derecha se atascaba antes de llegar a los 90° de inclinación.

3.3 Análisis de costos

Partiendo del estado inicial del proyecto, no se tomaron en cuenta el costo de la estructura ni de los motores. Solo se consideraron los elementos del sistema de control y sensores, menos la tarjeta Raspberry Pi y el Arduino Mega, que formaban parte del proyecto anterior. En la tabla 3.2 se muestra el desglose de los mismos.

Tabla 3.4. Análisis de costos del proyecto

[Elaboración propia]

Elemento	**Cantidad**	**Costo Unitario**	**Subtotal**
MPU-6050	6 módulos	$ 4	$ 24
Arduino UNO	3 tarjetas	$ 10	$ 30
USB-Hub 4 puertos	1 dispositivo	$ 5	$ 5
Baquelita de 10x10cm	1 placa	$1	$1
Extensión cables USB-Serial	3 cables	$3	$9
Cable UTP	4 metros	$1	$4
Jumpers	2 paquetes de 40	$4	$8
		Total	$81

Como puede observarse, el costo del proyecto no es alto y se debe en mayor parte al costo de los módulos MPU y las tarjetas Arduino empleadas en el mismo. Los MPU podrían conseguirse a menor precio importándolos, teniendo un costo en el mercado internacional de $2.50. Si se tomase en cuenta las tarjetas Arduino Mega y Raspberry, el costo total del proyecto ascendería alrededor de los $150.

CAPÍTULO 4

4. CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- La implementación de sensores inerciales mejoró la calidad de los coeficientes obtenidos durante la rutina de calibración y entrenamiento del exoesqueleto, lo que resultó en una ejecución más fluida del movimiento de caminata que con los coeficientes determinados en base a lecturas de los potenciómetros. Un análisis de Pearson reveló que la articulación de la cadera tuvo una mejora muy significativa en cuanto a la correlación de datos respecto a los valores deseados experimentalmente obtenidos.
- La estructura del cableado y ergonomía del exoesqueleto en su parte frontal mejoraron considerando que los sensores actualmente están instalados en la parte externa de la armadura. Sin embargo, el uso de microcontroladores Arduino hizo que el cableado posterior aumentase en volumen por el número de conexiones necesarias para establecer el protocolo SPI.
- Los acelerómetros también agilizaron el proceso de calibración al poder determinar por sí mismos la posición neutral (parado) respecto a la dirección de la gravedad, sin necesidad que un operador fije manualmente los límites de movimiento necesarios.
- Los sensores proporcionaron lecturas adecuadas incluso al no estar perfectamente alineados al plano vertical, lo cual permite la eventual implementación de grados de libertad laterales que simulen los movimientos de cambio de dirección al caminar.
- El bajo costo de los sensores y los microcontroladores hacen al proyecto bastante viable económicamente. Además, es fácilmente adaptable a situaciones que requieren una entrada similar de datos de posición, ya que los MPU pueden desmontarse fácilmente deshaciendo la unión soldada y colocándolos en otro lugar.

4.2 Recomendaciones

- La parte electrónica del proyecto puede miniaturizarse aún más si se remueven los componentes innecesarios de los Arduino UNO y se trabaja directamente con su procesador AVR32. En este caso ya no se necesitaría usar cables USB-serial, sino cualquier tipo de línea de 5V DC.
- El algoritmo de Python tiene espacio para mejoras en cuanto a la presentación de la interfaz con el usuario.
- Realizar un mantenimiento general a los motores. El acople del motor de la cadera izquierda se rompió durante la operación y gira en blanco. El motor de la cadera derecha tiene problemas para llegar a más de 45° hacia adelante. El motor del tobillo izquierdo presenta problemas de ruido al operar. Uno de los drivers está quemado y tiene que cambiarse.
- Considerar reemplazar el acople de los motores de las caderas por piezas maquinadas en acero. Estos motores son los que generan más torque y por ende todos los elementos que intervengan en la transmisión de potencia deberían tener una resistencia acorde al nivel de esfuerzos a soportar.
- Los drivers de los motores Escon/EC están optimizados para trabajar con sensores de efecto Hall, por lo cual debería considerarse la posibilidad de adquirir estos como complemento al sistema actual de sensores inerciales.
- Implementar en el algoritmo de Python una orden para recalibrar los MPU-6050 desde la Raspberry Pi. Actualmente solo se realiza subiendo un algoritmo desde la PC uno por uno en los Arduinos, y luego hay que volver a subir el algoritmo de recolección de datos.
- Diseñar un sistema para restringir el giro en el plano horizontal de los tobillos. Tienen juego excesivo lo cual hace que bailen durante la ejecución de las rutinas.
- Crear soportes dedicados para las tarjetas Arduino UNO.
- Reproducir el circuito de conexiones SPI y Arduino-Drivers en una pieza de baquelita para minimizar más el cableado y poder prescindir de los jumpers y de los protoboard.
- Implementar un algoritmo que permita usar los acelerómetros de los MPU para realizar un análisis de vibraciones a los elementos del exoesqueleto.

BIBLIOGRAFÍA

Begué, J. & Cobeña, W. (2017). Diseño y construcción de un prototipo de la estructura mecánica de un exoesqueleto para rehabilitación de niños con discapacidad motora en extremidades inferiores. Trabajo final para la obtención del título de Ing. Mecánico. Espol. FIMCP, Guayaquil.

Alcívar, E. (2018). Desarrollo de algoritmos de control para un exoesqueleto robótico de 6 GDL. Trabajo final para la obtención del título de Ing. Mecánico. Espol, FIMCP, Guayaquil.

Mármol, J. & Sánchez, G. (2019). Automatización de ciclos de movimiento programado para un exoesqueleto de 6-GDL. Trabajo final para la obtención del título de Ing. Mecánico. Espol. FIMCP, Guayaquil.

Bortole, M., & Pons, J. L. (2013). Development of an Exoskeleton for Lower Limb Rehabilitation. Biosystems & Biorobotics, 85–90. doi:10.1007/978-3-642-34546-3_14

Kruse, R., Borgelt, C., Klawonn, F., Moewes, C., Steinbrecher, M., & Held, P. (2013). Computational Intelligence. Texts in Computer Science. doi:10.1007/978-1-4471-5013-8

Young, A. J., Gannon, H., & Ferris, D. P. (2017). A Biomechanical Comparison of Proportional Electromyography Control to Biological Torque Control Using a Powered Hip Exoskeleton. Frontiers in Bioengineering and Biotechnology, 5. doi:10.3389/fbioe.2017.00037

Dollar, A. M., & Herr, H. (2008). Lower Extremity Exoskeletons and Active Orthoses: Challenges and State-of-the-Art. IEEE Transactions on Robotics, 24(1), 144–158. doi:10.1109/tro.2008.915453

Apostolyuk, V. (n.d.). Theory and Design of Micromechanical Vibratory Gyroscopes. MEMS/NEMS, 173–195. doi:10.1007/0-387-25786-1_6

Leens, F. (2009). An introduction to I2C and SPI protocols. IEEE Instrumentation & Measurement Magazine, 12(1), 8–13. doi:10.1109/mim.2009.4762946

Bovi, G., Rabuffetti, M., Mazzoleni, P., & Ferrarin, M. (2011). A multiple-task gait analysis approach: Kinematic, kinetic and EMG reference data for healthy young and adult subjects. Gait & Posture, 33(1), 6-13. doi: 10.1016/j.gaitpost.2010.08.009

Bhuyan, Ariful & Mallick, Tuton. (2014). Gyro-accelerometer based control of a robotic Arm using AVR microcontroller. 2014 9th International Forum on Strategic Technology, IFOST 2014. 409-413. 10.1109/IFOST.2014.6991151.

APÉNDICES

APÉNDICE A

Algoritmo de calibración de MPU6050 [Fuente: https://naylampmechatronics.com]

```
// calibrar_mpu6050.ino
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"

MPU6050 sensor;

// Valores RAW (sin procesar) del acelerometro y giroscopio en los ejes x,y,z
int ax, ay, az;
int gx, gy, gz;

//Variables usadas por el filtro pasa bajos
long f_ax,f_ay, f_az;
int p_ax, p_ay, p_az;
long f_gx,f_gy, f_gz;
int p_gx, p_gy, p_gz;
int counter=0;

//Valor de los offsets
int ax_o,ay_o,az_o;
int gx_o,gy_o,gz_o;

void setup() {
  Serial.begin(9600);   //Iniciando puerto serial
  Wire.begin();          //Iniciando I2C
  sensor.initialize();    //Iniciando el sensor

  if (sensor.testConnection()) Serial.println("Sensor iniciado correctamente");

  // Leer los offset los offsets anteriores
  ax_o=sensor.getXAccelOffset();
  ay_o=sensor.getYAccelOffset();
  az_o=sensor.getZAccelOffset();
  gx_o=sensor.getXGyroOffset();
  gy_o=sensor.getYGyroOffset();
  gz_o=sensor.getZGyroOffset();

  Serial.println("Offsets:");
```

```
  Serial.print(ax_o); Serial.print("\t");
  Serial.print(ay_o); Serial.print("\t");
  Serial.print(az_o); Serial.print("\t");
  Serial.print(gx_o); Serial.print("\t");
  Serial.print(gy_o); Serial.print("\t");
  Serial.print(gz_o); Serial.println("\t");

  Serial.println("nnEnvie cualquier caracter para empezar la calibracionnn");
  // Espera un caracter para empezar a calibrar
  while (true){if (Serial.available()) break;}
  Serial.println("Calibrando, no mover IMU");
}

void loop() {
  // Leer las aceleraciones y velocidades angulares
  sensor.getAcceleration(&ax, &ay, &az);
  sensor.getRotation(&gx, &gy, &gz);

  // Filtrar las lecturas
  f_ax = f_ax-(f_ax>>5)+ax;
  p_ax = f_ax>>5;

  f_ay = f_ay-(f_ay>>5)+ay;
  p_ay = f_ay>>5;

  f_az = f_az-(f_az>>5)+az;
  p_az = f_az>>5;

  f_gx = f_gx-(f_gx>>3)+gx;
  p_gx = f_gx>>3;

  f_gy = f_gy-(f_gy>>3)+gy;
  p_gy = f_gy>>3;

  f_gz = f_gz-(f_gz>>3)+gz;
  p_gz = f_gz>>3;
  //Cada 100 lecturas corregir el offset
  if (counter==100){
    //Mostrar las lecturas separadas por un [tab]
    Serial.print("promedio:"); Serial.print("\t");
```

```
    Serial.print(p_ax); Serial.print("\t");
    Serial.print(p_ay); Serial.print("\t");
    Serial.print(p_az); Serial.print("\t");
    Serial.print(p_gx); Serial.print("\t");
    Serial.print(p_gy); Serial.print("\t");
    Serial.println(p_gz);

    //Calibrar el acelerometro a 1g en el eje x (ajustar el offset)
    if (p_ax+16384<0) ax_o++;
    else {ax_o--;}
    if (p_ay>0) ay_o--;
    else {ay_o++;}
    if (p_az>0) az_o--;
    else {az_o++;}

    sensor.setXAccelOffset(ax_o);
    sensor.setYAccelOffset(ay_o);
    sensor.setZAccelOffset(az_o);

    if (p_gx>0) gx_o--;
    else {gx_o++;}
    if (p_gy>0) gy_o--;
    else {gy_o++;}
    if (p_gz>0) gz_o--;
    else {gz_o++;}

    sensor.setXGyroOffset(gx_o);
    sensor.setYGyroOffset(gy_o);
    sensor.setZGyroOffset(gz_o);

    counter=0;
  }
  counter++;
}
```

APÉNDICE B

Algoritmo de prueba para lectura de un MPU, enviando la información a un maestro por SPI [Elaboración propia]

```
#include<Wire.h>
#include<MPU6050.h>

const int mpuAddress = 0x68;
MPU6050 mpu(mpuAddress);

int ax, ay, az;
int gx, gy, gz;

long tiempo_prev;
float dt;
float ang_x, ang_y;
float ang_x_prev, ang_y_prev;

double z;

int a1;

volatile byte command = 0;
volatile byte b1, b2;

void updateFiltered()
{
  dt = (millis() - tiempo_prev) / 1000.0;
  tiempo_prev = millis();

  //Calcular los ángulos con acelerometro
  float accel_ang_x = atan(ay / sqrt(pow(ax, 2) + pow(az, 2)))*(180.0 / 3.1416);

  //Calcular angulo de rotación con giroscopio y filtro complementario
  ang_x = 0.98*(ang_x_prev + (gx / 131)*dt) + 0.02*accel_ang_x;

  ang_x_prev = ang_x;
  }
```

```
void setup (void)
  {

  // have to send on master in, *slave out*
  pinMode(MISO, OUTPUT);

  // turn on SPI in slave mode
  SPCR |= _BV(SPE);

  // turn on interrupts
  SPCR |= _BV(SPIE);

  Serial.begin(9600);
  Wire.begin();
  mpu.initialize();

  }  // end of setup

// SPI interrupt routine
ISR (SPI_STC_vect)
  {
  byte c = SPDR;

  switch (command)
    {
      case 0:
      command = c;
      SPDR = 0;
      break;

    case 'c':
        SPDR = c*b1 + (1-c)*b2;
         break;
      }// end of switch
}  // end of interrupt service routine (ISR) SPI_STC_vect

void loop (void)
  {
   mpu.getAcceleration(&ax, &ay, &az);
   mpu.getRotation(&gx, &gy, &gz);
```

```
 updateFiltered();

 a1=100*ang_x;
 b1=lowByte(a1);
 b2=highByte(a1);

 delay(10);
 Serial.println(ang_x);

if (digitalRead (SS) == HIGH)
  command = 0;
}  // end of loop
```

APÉNDICE C

Algoritmo final de Arduino para las rutinas de movimiento de la pierna derecha [Elaboración propia]:

```
// Notes:
// right ankle: set LOW (at direction) to ‘reduce’ angular position (negative displacement).
// right knee: set LOW (at direction) to ‘reduce’ angular position (negative displacement).
// right hip: set LOW (at direction) to ‘reduce’ angular position (negative displacement).

int rightAnkInit;
int rightKneeInit;
int rightHipInit;

int incomingByte;

int rightAnkPwmValues[101];
int rightKnePwmValues[101];
int rightHipPwmValues[101];



int i;
int ii;

////////////////////////// right leg. //////////////////////////
// right ankle:
int rightAnkEnable = 28;
int rightAnkDirection = 29;
int rightAnkPwm = 10;

int rightAnkPositionFeedback[100];
int rightAnkStartPosition;
int rightAnkPosition; //

int rightAnkRealPosition[100];
int rightAnkRealTime[100];

int rightAnkLoLimit = 0; //
int rightAnkHiLimit = 0; //
//
// right knee:
```

```
int rightKneEnable = 30;
int rightKneDirection = 31;
int rightKnePwm = 9;
int rightKnePosition;

int rightKnePositionFeedback[100];
int rightKneStartPosition;

int rightKneRealPosition[100];
int rightKneRealTime[100];

int rightKneLoLimit = 0; //
int rightKneHiLimit = 0; //
//
// right hip:
int rightHipEnable = 32;
int rightHipDirection = 33;
int rightHipPwm = 11;
int rightHipPosition;

int rightHipPositionFeedback[100];
int rightHipStartPosition;

int rightHipRealPosition[100];
int rightHipRealTime[100];

int rightHipLoLimit = 0; //
int rightHipHiLimit = 0; //
////////////////////////////////////////


int b = 0;
int eval[2];
float timeZero;
float timeTwo;
int limitVerification = 0;
int stOp;
////////////////////////////////////////
```

```
int inbetweenTime;

//////////////////////////////////////////

int driversEn[6];

//////////////////////////////////////////

int boundary;

//////////////////////////////////////////

int limitsDone = 0;
int loLimits[6];
int hiLimits[6];

//////////////////////////////////////////

int times;
int first;

//////////////////////////////////////////

int rightAnkWiseValue[100];
int rightKneWiseValue[100];
int rightHipWiseValue[100];

#include <SPI.h>
int ac1;
int ar1;
int at1;

byte hb1, lb1;

byte transferAndWait (const byte what)    //////definición de proceso de transferencia SPI/////
{
  byte a = SPI.transfer (what);
  delayMicroseconds (40);
  return a;
} // end of transferAndWait
```

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);

  ////////////////////////// right leg. //////////////////////////
  // right ankle:
  pinMode(rightAnkEnable, OUTPUT);
  pinMode(rightAnkDirection, OUTPUT);
  pinMode(rightAnkPwm, OUTPUT);
  // pinMode(rightAnkFeedbackPower, OUTPUT);

  //
  // right knee:
  pinMode(rightKneEnable, OUTPUT);
  pinMode(rightKneDirection, OUTPUT);
  pinMode(rightKnePwm, OUTPUT);
  // pinMode(rightKneFeedbackPower, OUTPUT);

  //
  // right hip:
  pinMode(rightHipEnable, OUTPUT);
  pinMode(rightHipDirection, OUTPUT);
  pinMode(rightHipPwm, OUTPUT);
  // pinMode(rightHipFeedbackPower, OUTPUT);
  //

  SPI.beginTransaction(SPISettings(4000000, MSBFIRST, SPI_MODE0));
  digitalWrite(SS, HIGH);  // ensure SS stays high for now
  pinMode(49, OUTPUT);
  digitalWrite(49, HIGH);
  SPI.begin ();

}

  ////////////////////////////////////////

  void loop() {
  // put your main code here, to run repeatedly:
  i=0;
```

```
digitalWrite(rightAnkEnable,LOW);
digitalWrite(rightKneEnable,LOW);
digitalWrite(rightHipEnable,LOW);


while (Serial.available() == 0){
  // Ingreso de primera tanda
}
// Primera tanda:
if (Serial.available() > 0) {////////////////////////// Main menu. //////////////////////////
  int incomingByte = Serial.parseInt();
  Serial.parseInt();

  if (incomingByte == 11111) {////////////////////////// Drivers enabling for training. //////////////////////////
    i=0;
    Serial.println(11);                                  // Confirmación.
    while (Serial.available() == 0){
       // Waiting for joint enabling selection
       ;
    }
    while (Serial.available() > 0) {
      driversEn[i] = Serial.parseInt();
      i++;
    }
    i=0;
    Serial.println(12);                                  // Confirmación.
    }


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    if (incomingByte == 22200) {
    rightAnkInit = 0;
    rightKneeInit = 0;
    rightHipInit = 0;


  digitalWrite(SS, LOW);
```

```
transferAndWait ('c');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(SS, HIGH);

ac1 = hb1;
ac1 = ac1 << 8;
ac1 |= lb1;

delay (100);



digitalWrite(49, LOW);

transferAndWait ('r');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(49, HIGH);

ar1 = hb1;              //send x_high to rightmost 8 bits
ar1 = ar1 << 8;       //shift x_high over to leftmost 8 bits
ar1 |= lb1;

delay (100);

digitalWrite(49, LOW);

transferAndWait ('t');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(49, HIGH);
```

```
at1 = hb1;              //send x_high to rightmost 8 bits
at1 = at1 << 8;       //shift x_high over to leftmost 8 bits
at1 |= lb1;

delay (100);


  rightAnkStartPosition  = at1;
  rightKneStartPosition  = ar1;
  rightHipStartPosition  = ac1;

  Serial.println("Posicion cadera: ");
  Serial.println(ac1);

  Serial.println("Posicion rodilla: ");
  Serial.println(ar1);



  Serial.println("Posicion tobillo: ");
  Serial.println(at1);

  driversEn[0]=1;
  driversEn[1]=1;
  driversEn[2]=1;

  ////////////////////////////////////////////////////VERIFICACION/////////////////////////////////////
  ////////////////////////////////////////////////////////CADERA/////////////////////////////////////////////////////////////////////////
    if (driversEn[2] == 1) {// right hip verif:
      if (rightHipStartPosition < rightHipInit) {
        digitalWrite(rightHipDirection,HIGH); ////////////////////////////////////////////////////////////////////////////
        analogWrite(rightHipPwm, 25);
        digitalWrite(rightHipEnable,HIGH);
        while (rightHipStartPosition < rightHipInit) {
          digitalWrite(SS, LOW);

transferAndWait ('c');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);
```

```
// disable Slave Select
digitalWrite(SS, HIGH);

ac1 = hb1;
ac1 = ac1 << 8;
ac1 |= lb1;


delay (100);
        rightHipStartPosition  = ac1;
          Serial.println("Posicion cadera: ");
  Serial.println(ac1);
       }
       analogWrite(rightHipPwm, 0);
       digitalWrite(rightHipEnable,LOW);
       digitalWrite(rightHipDirection,HIGH); ////////////////////////////////////////////////////////////////////////
     }
     else if (rightHipStartPosition > rightHipInit) {
       digitalWrite(rightHipDirection,LOW); ////////////////////////////////////////////////////////////////////////
       analogWrite(rightHipPwm, 25);
       digitalWrite(rightHipEnable,HIGH);
       while (rightHipStartPosition > rightHipInit) {
          digitalWrite(SS, LOW);

transferAndWait ('c');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(SS, HIGH);

ac1 = hb1;
ac1 = ac1 << 8;
ac1 |= lb1;


delay (100);
        rightHipStartPosition  = ac1;
```

```
          Serial.println("Posicion cadera: ");
  Serial.println(ac1);
       }
       analogWrite(rightHipPwm, 0);
       digitalWrite(rightHipEnable,LOW);
       digitalWrite(rightHipDirection,HIGH);
     }
     Serial.println("Cadera calibrada");
   }// End right hip verif.

 ////////////////////////////////////////////////////////RODILLA/////////////////////////////////////////////////////////////////


   if (driversEn[1] == 1) {// right knee verif:
     if (rightKneStartPosition < rightKneeInit) {
       digitalWrite(rightKneDirection,HIGH); ////////////////////////////////////////////////////////////////////////////
       analogWrite(rightKnePwm, 25);
       digitalWrite(rightKneEnable,HIGH);
       while (rightKneStartPosition < rightKneeInit) {
         digitalWrite(49, LOW);

transferAndWait ('r');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(49, HIGH);

ar1 = hb1;              //send x_high to rightmost 8 bits
ar1 = ar1 << 8;       //shift x_high over to leftmost 8 bits
ar1 |= lb1;

delay (100);  // 1 second delay
          rightKneStartPosition  = ar1;
          Serial.println("Posicion rodilla: ");
  Serial.println(ar1);
       }
       analogWrite(rightKnePwm, 0);
       digitalWrite(rightKneEnable,LOW);
```

```
      digitalWrite(rightKneDirection,HIGH); ////////////////////////////////////////////////////////////////////
    }
    else if (rightKneStartPosition > rightKneeInit) {
      digitalWrite(rightKneDirection,LOW); ////////////////////////////////////////////////////////////////////
      analogWrite(rightKnePwm, 25);
      digitalWrite(rightKneEnable,HIGH);
      while (rightKneStartPosition > rightKneeInit) {
        digitalWrite(49, LOW);

transferAndWait ('r');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(49, HIGH);

ar1 = hb1;              //send x_high to rightmost 8 bits
ar1 = ar1 << 8;       //shift x_high over to leftmost 8 bits
ar1 |= lb1;

delay (100);  // 1 second delay
         rightKneStartPosition  = ar1;
          Serial.println("Posicion rodilla: ");
  Serial.println(ar1);
       }
       analogWrite(rightKnePwm, 0);
       digitalWrite(rightKneEnable,LOW);
       digitalWrite(rightKneDirection,HIGH); ////////////////////////////////////////////////////////////////////
     }
    Serial.println("Rodilla calibrada");
   }// End right knee verif.

//////////////////////////////////////////////////////TOBILLO////////////////////////////////////////////////////

  if (driversEn[0] == 1) {// right ankle verif:
     if (rightAnkStartPosition < rightAnkInit) {
       digitalWrite(rightAnkDirection,HIGH); ////////////////////////////////////////////////////////////////////
       analogWrite(rightAnkPwm, 25);
       digitalWrite(rightAnkEnable,HIGH); ////////////////////////////////////////////////////////////////////
```

```
      while (rightAnkStartPosition < rightAnkInit) {
        digitalWrite(49, LOW);

        transferAndWait ('t');  // add command
        transferAndWait (0);
        hb1 = transferAndWait (1);
        lb1 = transferAndWait (0);

        // disable Slave Select
        digitalWrite(49, HIGH);

       at1 = hb1;              //send x_high to rightmost 8 bits
       at1 = at1 << 8;       //shift x_high over to leftmost 8 bits
       at1 |= lb1;
       delay (100);
       rightAnkStartPosition  = at1;
       Serial.println("Posicion tobillo: ");
Serial.println(at1);
      }
      analogWrite(rightAnkPwm, 0);
      digitalWrite(rightAnkEnable,LOW);
      digitalWrite(rightAnkDirection,HIGH); ////////////////////////////////////////////////////////////////////////
    }
    else if (rightAnkStartPosition > rightAnkInit) {
      digitalWrite(rightAnkDirection,LOW); ////////////////////////////////////////////////////////////////////////
      analogWrite(rightAnkPwm, 25);
      digitalWrite(rightAnkEnable,HIGH);
      while (rightAnkStartPosition > rightAnkInit) {

        digitalWrite(49, LOW);

        transferAndWait ('t');  // add command
        transferAndWait (0);
        hb1 = transferAndWait (1);
        lb1 = transferAndWait (0);

        // disable Slave Select
        digitalWrite(49, HIGH);

       at1 = hb1;              //send x_high to rightmost 8 bits
```

```
          at1 = at1 << 8;       //shift x_high over to leftmost 8 bits
          at1 |= lb1;
          delay (100);
           rightAnkStartPosition  = at1;
            Serial.println("Posicion tobillo: ");
    Serial.println(at1);
         }
         analogWrite(rightAnkPwm, 0);
         digitalWrite(rightAnkEnable,LOW);
         digitalWrite(rightAnkDirection,HIGH); ////////////////////////////////////////////////////////////////////////

        }
      Serial.println("Tobillo Calibrado");
      }// End right ankle verif.



      Serial.println("Calibración terminada");
      delay(10000);
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    if (incomingByte == 11000) {////////////////////////// Free movement. //////////////////////////

      Serial.println(2);                                    // Confirmación.
      while (Serial.available() == 0){
        // Waiting for joint selection
        ;
      }
      // Joint selection:
      if (Serial.available() > 0) {
        int incomingByte = Serial.parseInt();
        Serial.parseInt();
        if (incomingByte == 11050) { // right ankle movement:
          digitalWrite(49, LOW);

            transferAndWait ('t');  // add command
            transferAndWait (0);
            hb1 = transferAndWait (1);
            lb1 = transferAndWait (0);
```

```
  // disable Slave Select
  digitalWrite(49, HIGH);

 at1 = hb1;            //send x_high to rightmost 8 bits
 at1 = at1 << 8;       //shift x_high over to leftmost 8 bits
 at1 |= lb1;
 delay (100);

rightAnkStartPosition  = at1;
Serial.println(201);                                  // Confirmación.
while (Serial.available() == 0) {
  // Ingreso de primera tanda
  ;
}
while (Serial.available() > 0) {
  int incomingByte = Serial.parseInt();
  rightAnkPwmValues[i] = incomingByte;
  i++;
}
Serial.println(202);                                  // Confirmación.
while (Serial.available() == 0){
  // Ingreso de delay
  ;
}
if (Serial.available() > 0) {
  inbetweenTime = Serial.parseInt();
  Serial.parseInt();
  for (int j=0; j < i-1; j++){
    if (rightAnkPwmValues[j] < 0) {
      digitalWrite(rightAnkDirection,LOW);
      rightAnkPwmValues[j]=-rightAnkPwmValues[j];
    }
    else if (rightAnkPwmValues[j] > 0) {
      digitalWrite(rightAnkDirection,HIGH);
    }
    digitalWrite(rightAnkEnable,HIGH);
    analogWrite(rightAnkPwm, rightAnkPwmValues[j]);
    delay(inbetweenTime);
    analogWrite(rightAnkPwm, 0);
    digitalWrite(rightAnkEnable,LOW);
```

```
          digitalWrite(49, LOW);

         transferAndWait ('t');  // add command
         transferAndWait (0);
         hb1 = transferAndWait (1);
         lb1 = transferAndWait (0);

         // disable Slave Select
         digitalWrite(49, HIGH);

        at1 = hb1;              //send x_high to rightmost 8 bits
        at1 = at1 << 8;       //shift x_high over to leftmost 8 bits
        at1 |= lb1;
        delay (100);
          rightAnkPositionFeedback[j] = at1;
         }
         Serial.println(rightAnkStartPosition);
         for (int j=0; j < i-1; j++){
          Serial.println(rightAnkPositionFeedback[j]);
         }
       }
     }
     else if (incomingByte == 11051) { // right knee movement:
       digitalWrite(49, LOW);

transferAndWait ('r');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(49, HIGH);

ar1 = hb1;              //send x_high to rightmost 8 bits
ar1 = ar1 << 8;       //shift x_high over to leftmost 8 bits
ar1 |= lb1;

delay (100);  // 1 second delay
       rightKneStartPosition  = ar1;
       Serial.println(211);                                    // Confirmación.
```

```
      while (Serial.available() == 0) {
        // Ingreso de primera tanda
        ;
      }
      while (Serial.available() > 0) {
        int incomingByte = Serial.parseInt();
        rightKnePwmValues[i] = incomingByte;
        i++;
      }
      Serial.println(212);                                    // Confirmación."Listo data.");
      while (Serial.available() == 0){
        // Ingreso de delay
        ;
      }
      if (Serial.available() > 0) {
        inbetweenTime = Serial.parseInt();
        Serial.parseInt();
        for (int j=0; j < i-1; j++){
          if (rightKnePwmValues[j] < 0) {
            digitalWrite(rightKneDirection,LOW);
            rightKnePwmValues[j]=-rightKnePwmValues[j];
          }
          else if (rightKnePwmValues[j] > 0) {
            digitalWrite(rightKneDirection,HIGH);
          }
          digitalWrite(rightKneEnable,HIGH);
          analogWrite(rightKnePwm, rightKnePwmValues[j]);
          delay(inbetweenTime);
          analogWrite(rightKnePwm, 0);
          digitalWrite(rightKneEnable,LOW);
          digitalWrite(49, LOW);

transferAndWait ('r');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(49, HIGH);
```

```
ar1 = hb1;              //send x_high to rightmost 8 bits
ar1 = ar1 << 8;       //shift x_high over to leftmost 8 bits
ar1 |= lb1;

delay (100);  // 1 second delay
          rightKnePositionFeedback[j] = ar1;
         }
         Serial.println(rightKneStartPosition);
         for (int j=0; j < i-1; j++){
          Serial.println(rightKnePositionFeedback[j]);
         }
       }
     }
     else if (incomingByte == 11052) { // right hip movement:
        digitalWrite(SS, LOW);

transferAndWait ('c');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(SS, HIGH);

ac1 = hb1;
ac1 = ac1 << 8;
ac1 |= lb1;



delay (100);
       rightHipStartPosition  = ac1;
       Serial.println(221);                                    // Confirmación."right hip ready.");
       while (Serial.available() == 0) {
         // Ingreso de primera tanda
         ;
       }
       while (Serial.available() > 0) {
         int incomingByte = Serial.parseInt();
         rightHipPwmValues[i] = incomingByte;
         i++;
```

```
      }
      Serial.println(222);                                  // Confirmación."Listo data.");
      while (Serial.available() == 0){
        // Ingreso de delay
        ;
      }
      if (Serial.available() > 0) {
        inbetweenTime = Serial.parseInt();
        Serial.parseInt();
        for (int j=0; j < i-1; j++){
          if (rightHipPwmValues[j] < 0) {
            digitalWrite(rightHipDirection,LOW);
            rightHipPwmValues[j]=-rightHipPwmValues[j];
          }
          else if (rightHipPwmValues[j] > 0) {
            digitalWrite(rightHipDirection,HIGH);
          }
          digitalWrite(rightHipEnable,HIGH);
          analogWrite(rightHipPwm, rightHipPwmValues[j]);
          delay(inbetweenTime);
          analogWrite(rightHipPwm, 0);
          digitalWrite(rightHipEnable,LOW);
           digitalWrite(SS, LOW);

transferAndWait ('c');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(SS, HIGH);

ac1 = hb1;
ac1 = ac1 << 8;
ac1 |= lb1;



delay (100);
          rightHipPositionFeedback[j] = ac1;
        }
```

```
      Serial.println(rightHipStartPosition);
      for (int j=0; j < i-1; j++){
        Serial.println(rightHipPositionFeedback[j]);
      }
    }
  }



////////////////////////// End free movement. //////////////////////////

else if (incomingByte == 11100) {////////////////////////// right leg factors. //////////////////////////


  Serial.println(3);                                    //Confirmación

  while (Serial.available() == 0){
    // Waiting for leg selection
    ;
  }
  // Leg selection:
  if (Serial.available() > 0) {
    int incomingByte = Serial.parseInt();
    Serial.parseInt();

    int boundary = 0;

    if (incomingByte == 11150) { // right ankle factor:
      Serial.println(301);
      while (Serial.available() == 0){
        // Waiting for boundary...
        ;
      }
      if (Serial.available() > 0) {
        boundary = Serial.parseInt();
      }
      digitalWrite(49, LOW);

        transferAndWait ('t');  // add command
        transferAndWait (0);
```

```
  hb1 = transferAndWait (1);
  lb1 = transferAndWait (0);

  // disable Slave Select
  digitalWrite(49, HIGH);

 at1 = hb1;              //send x_high to rightmost 8 bits
 at1 = at1 << 8;       //shift x_high over to leftmost 8 bits
 at1 |= lb1;
 delay (100);
eval[0] = at1;

if (eval[0] > boundary) {    ////////////////////////////////////////////////////////////////////////////////////////////////////////////
  digitalWrite(rightAnkDirection,HIGH); ////////////////////////////////////
  analogWrite(rightAnkPwm, 50);
  digitalWrite(rightAnkEnable,HIGH);
  while (eval[0] > boundary) {
    digitalWrite(49, LOW);

  transferAndWait ('t');  // add command
  transferAndWait (0);
  hb1 = transferAndWait (1);
  lb1 = transferAndWait (0);

  // disable Slave Select
  digitalWrite(49, HIGH);

 at1 = hb1;              //send x_high to rightmost 8 bits
 at1 = at1 << 8;       //shift x_high over to leftmost 8 bits
 at1 |= lb1;
 delay (100);
    eval[0]=at1;
  }
  analogWrite(rightAnkPwm, 0);
  digitalWrite(rightAnkEnable,LOW);
  digitalWrite(rightAnkDirection,HIGH);////////////////////////////////////
  Serial.parseInt();
} // Novo
else if (eval[0] < boundary) {   ////////////////////////////////////////////////////////////////////////////////////////////////////////////
  digitalWrite(rightAnkDirection,LOW);////////////////////////////////////
```

```
 analogWrite(rightAnkPwm, 50);
 digitalWrite(rightAnkEnable,HIGH);
 while (eval[0] < boundary) {
   digitalWrite(49, LOW);

 transferAndWait ('t');  // add command
 transferAndWait (0);
 hb1 = transferAndWait (1);
 lb1 = transferAndWait (0);

 // disable Slave Select
 digitalWrite(49, HIGH);

 at1 = hb1;              //send x_high to rightmost 8 bits
 at1 = at1 << 8;       //shift x_high over to leftmost 8 bits
 at1 |= lb1;
 delay (100);
    eval[0]=at1;
 }
 analogWrite(rightAnkPwm, 0);
 digitalWrite(rightAnkEnable,LOW);
 digitalWrite(rightAnkDirection,HIGH);////////////
 Serial.parseInt();
}
digitalWrite(rightAnkDirection,LOW);////////////////////////////////////
digitalWrite(rightAnkEnable,HIGH);
analogWrite(rightAnkPwm, 100);
delay(1500);
digitalWrite(rightAnkEnable,LOW);
delay(500);
digitalWrite(49, LOW);

 transferAndWait ('t');  // add command
 transferAndWait (0);
 hb1 = transferAndWait (1);
 lb1 = transferAndWait (0);

 // disable Slave Select
 digitalWrite(49, HIGH);
```

```
 at1 = hb1;            //send x_high to rightmost 8 bits
 at1 = at1 << 8;       //shift x_high over to leftmost 8 bits
 at1 |= lb1;
 delay (100);
eval[1] = at1;
analogWrite(rightAnkPwm, 0);
Serial.println(eval[0]);
Serial.println(eval[1]);
digitalWrite(49, LOW);

  transferAndWait ('t');  // add command
  transferAndWait (0);
  hb1 = transferAndWait (1);
  lb1 = transferAndWait (0);

  // disable Slave Select
  digitalWrite(49, HIGH);

 at1 = hb1;            //send x_high to rightmost 8 bits
 at1 = at1 << 8;       //shift x_high over to leftmost 8 bits
 at1 |= lb1;
 delay (100);
eval[0] = at1;
if (eval[0] > boundary) {
  digitalWrite(rightAnkDirection,HIGH); ////////////////////////////////////
  analogWrite(rightAnkPwm, 50);
  digitalWrite(rightAnkEnable,HIGH);
  while (eval[0] > boundary) {
    digitalWrite(49, LOW);

transferAndWait ('t');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(49, HIGH);

at1 = hb1;            //send x_high to rightmost 8 bits
at1 = at1 << 8;       //shift x_high over to leftmost 8 bits
```

```
at1 |= lb1;

delay (100);
           eval[0]=at1;//;
         }
         analogWrite(rightAnkPwm, 0);
         digitalWrite(rightAnkEnable,LOW);
         digitalWrite(rightAnkDirection,HIGH);////////////////////////////////////
       }

       //// End ankle factor. ///
     }
     else if (incomingByte == 11151) { // right knee factor:
       Serial.println(311);
       while (Serial.available() == 0){
         // Waiting for boundary...
         ;
       }
       if (Serial.available() > 0) {
         boundary = Serial.parseInt();
       }
       digitalWrite(49, LOW);

transferAndWait ('r');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(49, HIGH);

ar1 = hb1;              //send x_high to rightmost 8 bits
ar1 = ar1 << 8;       //shift x_high over to leftmost 8 bits
ar1 |= lb1;

delay (100);  // 1 second delay
       eval[0] = ar1;

       if (eval[0] > boundary) {
         digitalWrite(rightKneDirection,HIGH); ////////////////////////////////////
```

```
          analogWrite(rightKnePwm, 50);
          digitalWrite(rightKneEnable,HIGH);
          while (eval[0] > boundary) {
            digitalWrite(49, LOW);

transferAndWait ('r');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(49, HIGH);

ar1 = hb1;              //send x_high to rightmost 8 bits
ar1 = ar1 << 8;       //shift x_high over to leftmost 8 bits
ar1 |= lb1;

delay (100);
            eval[0]=ar1;//;
          }
          analogWrite(rightKnePwm, 0);
          digitalWrite(rightKneEnable,LOW);
          digitalWrite(rightKneDirection,HIGH); ////////////////////////////////////
          Serial.parseInt();
        } // Novo
        else if (eval[0] < boundary) {
          digitalWrite(rightKneDirection,LOW);////////////////////////////////////
          analogWrite(rightKnePwm, 50);
          digitalWrite(rightKneEnable,HIGH);
          while (eval[0] < boundary) {
            digitalWrite(49, LOW);

transferAndWait ('r');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(49, HIGH);
```

```
ar1 = hb1;              //send x_high to rightmost 8 bits
ar1 = ar1 << 8;       //shift x_high over to leftmost 8 bits
ar1 |= lb1;

delay (100);
          eval[0]=ar1;//;
         }
         analogWrite(rightKnePwm, 0);
         digitalWrite(rightKneEnable,LOW);
         digitalWrite(rightKneDirection,HIGH);////////////////
         Serial.parseInt();
       }
       digitalWrite(rightKneDirection,LOW); ////////////////////////////////////
       digitalWrite(rightKneEnable,HIGH);
       analogWrite(rightKnePwm, 100);
       delay(1500);
       digitalWrite(rightKneEnable,LOW);
       delay(500);
       digitalWrite(49, LOW);

transferAndWait ('r');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(49, HIGH);

ar1 = hb1;              //send x_high to rightmost 8 bits
ar1 = ar1 << 8;       //shift x_high over to leftmost 8 bits
ar1 |= lb1;

delay (100);
       eval[1] = ar1;
       analogWrite(rightKnePwm, 0);
       Serial.println(eval[0]);
       Serial.println(eval[1]);

       eval[0] = ar1;
       if (eval[0] > boundary) {
```

```
          digitalWrite(rightKneDirection,HIGH); ////////////////////////////////////
          analogWrite(rightKnePwm, 50);
          digitalWrite(rightKneEnable,HIGH);
          while (eval[0] > boundary) {
            digitalWrite(49, LOW);

transferAndWait ('r');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(49, HIGH);

ar1 = hb1;              //send x_high to rightmost 8 bits
ar1 = ar1 << 8;       //shift x_high over to leftmost 8 bits
ar1 |= lb1;

delay (100);  // 1 second delay
             eval[0]=ar1;//;
          }
          analogWrite(rightKnePwm, 0);
          digitalWrite(rightKneEnable,LOW);
          digitalWrite(rightKneDirection,HIGH); ////////////////////////////////////
        }

        //// End knee factor. ///
      }
      else if (incomingByte == 11152) { // right hip factor:
        Serial.println(321);
        while (Serial.available() == 0){
          // Waiting for boundary...
          ;
        }
        if (Serial.available() > 0) {
          boundary = Serial.parseInt();
        }
         digitalWrite(SS, LOW);

transferAndWait ('c');  // add command
```

```
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(SS, HIGH);

ac1 = hb1;
ac1 = ac1 << 8;
ac1 |= lb1;



delay (100);
      eval[0] = ac1;

      if (eval[0] > boundary) {
        digitalWrite(rightHipDirection,HIGH); ////////////////////////////////////
        analogWrite(rightHipPwm, 50);
        digitalWrite(rightHipEnable,HIGH);
        while (eval[0] > boundary) {
          digitalWrite(SS, LOW);

transferAndWait ('c');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(SS, HIGH);

ac1 = hb1;
ac1 = ac1 << 8;
ac1 |= lb1;



delay (100);
          eval[0]=ac1;//;
        }
        analogWrite(rightHipPwm, 0);
        digitalWrite(rightHipEnable,LOW);
```

```
        digitalWrite(rightHipDirection,HIGH); ////////////////////////////////////
        Serial.parseInt();
      } // Novo
      else if (eval[0] < boundary) {
        digitalWrite(rightHipDirection,LOW); ////////////////////////////////////
        analogWrite(rightHipPwm, 50);
        digitalWrite(rightHipEnable,HIGH);
        while (eval[0] < boundary) {
          digitalWrite(SS, LOW);

transferAndWait ('c');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(SS, HIGH);

ac1 = hb1;
ac1 = ac1 << 8;
ac1 |= lb1;


delay (100);
          eval[0]=ac1;//;
        }
        analogWrite(rightHipPwm, 0);
        digitalWrite(rightHipEnable,LOW);
        digitalWrite(rightHipDirection,HIGH);////////////////////////////////////
        Serial.parseInt();
      }

      digitalWrite(rightHipDirection,HIGH); ////////////////////////////////////
      digitalWrite(rightHipEnable,HIGH);
      analogWrite(rightHipPwm, 100);

      delay(1500);
      digitalWrite(rightHipEnable,LOW);
      delay(500);
       digitalWrite(SS, LOW);
```

```
transferAndWait ('c');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(SS, HIGH);

ac1 = hb1;
ac1 = ac1 << 8;
ac1 |= lb1;



delay (100);
      eval[1] = ac1;
      analogWrite(rightHipPwm, 0);
      Serial.println(eval[0]);
      Serial.println(eval[1]);

      eval[0] = ac1;
      if (eval[0] < boundary) {
        digitalWrite(rightHipDirection,LOW); ////////////////////////////////////
        analogWrite(rightHipPwm, 50);
        digitalWrite(rightHipEnable,HIGH);
        while (eval[0] < boundary) {
          digitalWrite(SS, LOW);

transferAndWait ('c');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(SS, HIGH);

ac1 = hb1;
ac1 = ac1 << 8;
ac1 |= lb1;
```

```
  delay (100);
            eval[0]= ac1;//;
          }
          analogWrite(rightHipPwm, 0);
          digitalWrite(rightHipEnable,LOW);
          digitalWrite(rightHipDirection,HIGH); ////////////////////////////////////
        }

        //// End right hip factor. ///
      }

        if (incomingByte == 111000) { // Quit:
          ;
        }
      }
    }
  }

   ////////////////////////////////// End factors. ////////////////////////////////


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////77
   else if (incomingByte == 11200) {////////////////////////// Training start. //////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

     rightAnkInit = 0;
   rightKneeInit = 0;
   rightHipInit = 0;

    digitalWrite(SS, LOW);

  transferAndWait ('c');  // add command
  transferAndWait (0);
  hb1 = transferAndWait (1);
  lb1 = transferAndWait (0);

  // disable Slave Select
```

```
digitalWrite(SS, HIGH);

ac1 = hb1;
ac1 = ac1 << 8;
ac1 |= lb1;

delay (100);

digitalWrite(49, LOW);

transferAndWait ('r');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(49, HIGH);

ar1 = hb1;              //send x_high to rightmost 8 bits
ar1 = ar1 << 8;       //shift x_high over to leftmost 8 bits
ar1 |= lb1;

delay (100);

digitalWrite(49, LOW);

transferAndWait ('t');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(49, HIGH);

at1 = hb1;              //send x_high to rightmost 8 bits
at1 = at1 << 8;       //shift x_high over to leftmost 8 bits
at1 |= lb1;

delay (100);
  rightAnkStartPosition  = at1;
```

```
rightKneStartPosition  = ar1;
rightHipStartPosition  = ac1;

driversEn[0]=1;
driversEn[1]=1;
driversEn[2]=1;

//////////////////////////////////////////////////VERIFICACION/////////////////////////////////
if (driversEn[0] == 1) {// right ankle verif:
   if (rightAnkStartPosition > rightAnkInit) {
     digitalWrite(rightAnkDirection,HIGH); ////////////////////////////////////////////////////////////////////////
     analogWrite(rightAnkPwm, 50);
     digitalWrite(rightAnkEnable,HIGH); ///////////////////////////////////////////////////////////////////////
     while (rightAnkStartPosition > rightAnkInit) {
       digitalWrite(49, LOW);

       transferAndWait ('t');  // add command
       transferAndWait (0);
       hb1 = transferAndWait (1);
       lb1 = transferAndWait (0);

       // disable Slave Select
       digitalWrite(49, HIGH);

      at1 = hb1;              //send x_high to rightmost 8 bits
      at1 = at1 << 8;       //shift x_high over to leftmost 8 bits
      at1 |= lb1;
      delay (100);
       rightAnkStartPosition  = at1;
     }
     analogWrite(rightAnkPwm, 0);
     digitalWrite(rightAnkEnable,LOW);
     digitalWrite(rightAnkDirection,HIGH); ////////////////////////////////////////////////////////////////////////
   }
   else if (rightAnkStartPosition < rightAnkInit) {
     digitalWrite(rightAnkDirection,LOW); ////////////////////////////////////////////////////////////////////////
     analogWrite(rightAnkPwm, 50);
     digitalWrite(rightAnkEnable,HIGH);
     while (rightAnkStartPosition < rightAnkInit) {
       digitalWrite(49, LOW);
```

```
        transferAndWait ('t');  // add command
        transferAndWait (0);
        hb1 = transferAndWait (1);
        lb1 = transferAndWait (0);

        // disable Slave Select
        digitalWrite(49, HIGH);

       at1 = hb1;              //send x_high to rightmost 8 bits
       at1 = at1 << 8;       //shift x_high over to leftmost 8 bits
       at1 |= lb1;
       delay (100);
        rightAnkStartPosition  = at1;
      }
      analogWrite(rightAnkPwm, 0);
      digitalWrite(rightAnkEnable,LOW);
      digitalWrite(rightAnkDirection,HIGH); ////////////////////////////////////////////////////////////////////////

    }
  }// End right ankle verif.

  if (driversEn[1] == 1) {// right knee verif:
    if (rightKneStartPosition > rightKneeInit) {
      digitalWrite(rightKneDirection,HIGH); ////////////////////////////////////////////////////////////////////////
      analogWrite(rightKnePwm, 50);
      digitalWrite(rightKneEnable,HIGH);
      while (rightKneStartPosition > rightKneeInit) {
        digitalWrite(49, LOW);

transferAndWait ('r');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(49, HIGH);

ar1 = hb1;              //send x_high to rightmost 8 bits
ar1 = ar1 << 8;       //shift x_high over to leftmost 8 bits
```

```
ar1 |= lb1;

delay (100);  // 1 second delay
          rightKneStartPosition  = ar1;
        }
        analogWrite(rightKnePwm, 0);
        digitalWrite(rightKneEnable,LOW);
        digitalWrite(rightKneDirection,HIGH); ////////////////////////////////////////////////////////////////////////
      }
      else if (rightKneStartPosition < rightKneeInit) {
        digitalWrite(rightKneDirection,LOW); ////////////////////////////////////////////////////////////////////////
        analogWrite(rightKnePwm, 50);
        digitalWrite(rightKneEnable,HIGH);
        while (rightKneStartPosition < rightKneeInit) {
          rightKneStartPosition  = ar1;
        }
        analogWrite(rightKnePwm, 0);
        digitalWrite(rightKneEnable,LOW);
        digitalWrite(rightKneDirection,HIGH); ////////////////////////////////////////////////////////////////////////
      }
    }// End right knee verif.

    if (driversEn[2] == 1) {// right hip verif:
      if (rightHipStartPosition > rightHipInit) {
        digitalWrite(rightHipDirection,HIGH); ////////////////////////////////////////////////////////////////////////
        analogWrite(rightHipPwm, 50);
        digitalWrite(rightHipEnable,HIGH);
        while (rightHipStartPosition > rightHipInit) {
           digitalWrite(SS, LOW);

transferAndWait ('c');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(SS, HIGH);

ac1 = hb1;
ac1 = ac1 << 8;
```

```
ac1 |= lb1;


delay (100);
          rightHipStartPosition  = ac1;
        }
        analogWrite(rightHipPwm, 0);
        digitalWrite(rightHipEnable,LOW);
        digitalWrite(rightHipDirection,HIGH); ////////////////////////////////////////////////////////////////////////
      }
      else if (rightHipStartPosition < rightHipInit) {
        digitalWrite(rightHipDirection,LOW); ////////////////////////////////////////////////////////////////////////
        analogWrite(rightHipPwm, 50);
        digitalWrite(rightHipEnable,HIGH);
        while (rightHipStartPosition < rightHipInit) {
           digitalWrite(SS, LOW);

transferAndWait ('c');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(SS, HIGH);

ac1 = hb1;
ac1 = ac1 << 8;
ac1 |= lb1;


delay (100);
          rightHipStartPosition  = ac1;
        }
        analogWrite(rightHipPwm, 0);
        digitalWrite(rightHipEnable,LOW);
        digitalWrite(rightHipDirection,HIGH); ////////////////////////////////////////////////////////////////////////
      }
    }// End right hip verif.

  ////////////////////////////////////////////////////////////////////////////////////////
```

```
analogWrite(rightAnkPwm, 0);
analogWrite(rightKnePwm, 0);
analogWrite(rightHipPwm, 0);

////////////////////////////////////PARTE 1/////////////////////////////////////
digitalWrite(rightKneEnable,HIGH);
digitalWrite(rightHipEnable,HIGH);

digitalWrite(rightKneDirection,LOW);
digitalWrite(rightHipDirection,HIGH);

analogWrite(rightKnePwm, 250);
analogWrite(rightHipPwm, 250);


delay(2750);
analogWrite(rightKnePwm, 0);
analogWrite(rightHipPwm, 0);

digitalWrite(rightHipEnable,LOW);
digitalWrite(rightKneEnable,LOW);

delay(2000);
////////////////////////////////////PARTE 2/////////////////////////////////////
digitalWrite(rightKneEnable,HIGH);
digitalWrite(rightHipEnable,HIGH);


digitalWrite(rightKneDirection,HIGH);
digitalWrite(rightHipDirection,LOW);


analogWrite(rightKnePwm, 250);
analogWrite(rightHipPwm, 250);


delay(2750);
analogWrite(rightKnePwm, 0);
analogWrite(rightHipPwm, 0);
```

```
  digitalWrite(rightHipEnable,LOW);
  digitalWrite(rightKneEnable,LOW);

 Serial.println(1);



 }
 else if (incomingByte == 11300) {////////////////////////// Movement execution. //////////////////////////

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////7

  Serial.println(52);
  i = 0;
  while (Serial.available() == 0){
    // Wainting reponse for data acq...
    ;
  }
  // Data acquisition:

  if (driversEn[0] == 1) {// right ankle data acq:
    while (Serial.available() > 0) {
      int incomingByte = Serial.parseInt();
      rightAnkPwmValues[i] = incomingByte;
      i++;
    }
    ii=i;
    i = 0;
    Serial.println(5201);
    // End right ankle.

    ///////////----------///////////
    while (Serial.available() == 0){
      // Wainting reponse...
      ;
    }
    ///////////----------///////////
  }
```

```
else {
  for (int j=0; j < 100; j++) {
    rightAnkPwmValues[j] = 0;
  }
}

if (driversEn[1] == 1) {// right knee data acq:
  while (Serial.available() > 0) {
    int incomingByte = Serial.parseInt();
    rightKnePwmValues[i] = incomingByte;
    i++;
  }
  ii=i;
  i = 0;

  Serial.println(5211);
  // End right knee.

  ///////////----------//////////
  while (Serial.available() == 0){
    // Wainting reponse...
    ;
  }
  ///////////----------//////////
}
else {
  for (int j=0; j < 100; j++) {
    rightKnePwmValues[j] = 0;
  }
}

if (driversEn[2] == 1) {// right hip data acq:
  while (Serial.available() > 0) {
    int incomingByte = Serial.parseInt();
    rightHipPwmValues[i] = incomingByte;
    i++;
  }
  ii=i;
  i = 0;
  Serial.println(5221);
```

```
    // End right hip.

    ///////////---------------------////////////
    while (Serial.available() == 0){
      // Wainting reponse...
      ;
    }
    ///////////---------------------////////////
  }
  else {
    for (int j=0; j < 100; j++) {
      rightHipPwmValues[j] = 0;
    }
  }



  ///////////////////////////////// End data acquisition. /////////////////////////////////
////////////////////////////////////////////%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%//////////////
/////////////////
  // Begin eval:
  while (Serial.available() == 0){
    // Ingreso de delay
    ;
  }
  if (Serial.available() > 0) {
    inbetweenTime = Serial.parseInt();
  }
  Serial.parseInt();

  Serial.println(5200);

  while (Serial.available() == 0){
    // Ingreso de número de repeticiones
    ;
  }
  if (Serial.available() > 0) {
    times = Serial.parseInt();
  }
```

```
    Serial.parseInt();

    ///// Verif initial position:

    ////////////////////// right leg position verification. //////////////////////

     digitalWrite(SS, LOW);

transferAndWait ('c');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(SS, HIGH);

ac1 = hb1;
ac1 = ac1 << 8;
ac1 |= lb1;

delay (100);

digitalWrite(49, LOW);

transferAndWait ('r');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(49, HIGH);

ar1 = hb1;              //send x_high to rightmost 8 bits
ar1 = ar1 << 8;       //shift x_high over to leftmost 8 bits
ar1 |= lb1;

delay (100);

digitalWrite(49, LOW);
```

```
transferAndWait ('t');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(49, HIGH);

at1 = hb1;              //send x_high to rightmost 8 bits
at1 = at1 << 8;       //shift x_high over to leftmost 8 bits
at1 |= lb1;

delay (100);

   rightAnkStartPosition = at1;
   rightKneStartPosition = ar1;
   rightHipStartPosition = ac1;
   ii=ii+1;
   if (driversEn[0] == 1) {// right ankle verif:
    /* Serial.println(rightAnkStartPosition);
     Serial.println(ii);
     Serial.println(rightAnkPwmValues[ii-2]);

     while (Serial.available() == 0){
    //
    ;
   }*/
    if (rightAnkStartPosition > rightAnkPwmValues[ii-2]) {
      digitalWrite(rightAnkDirection,HIGH); ////////////////////////////////////////////////////////////////////////
      analogWrite(rightAnkPwm, 50);
      digitalWrite(rightAnkEnable,HIGH); ////////////////////////////////////////////////////////////////////////
      while (rightAnkStartPosition > rightAnkPwmValues[ii-2]) {
        digitalWrite(49, LOW);

        transferAndWait ('t');  // add command
        transferAndWait (0);
        hb1 = transferAndWait (1);
        lb1 = transferAndWait (0);

        // disable Slave Select
```

```
    digitalWrite(49, HIGH);

    at1 = hb1;              //send x_high to rightmost 8 bits
    at1 = at1 << 8;         //shift x_high over to leftmost 8 bits
    at1 |= lb1;
    delay (100);
     rightAnkStartPosition  = at1;
   }
   analogWrite(rightAnkPwm, 0);
   digitalWrite(rightAnkEnable,LOW);
   digitalWrite(rightAnkDirection,HIGH); ////////////////////////////////////////////////////////////////////////
  }
  else if (rightAnkStartPosition < rightAnkPwmValues[ii-2]) {
   digitalWrite(rightAnkDirection,LOW); ////////////////////////////////////////////////////////////////////////
   analogWrite(rightAnkPwm, 50);
   digitalWrite(rightAnkEnable,HIGH);
   while (rightAnkStartPosition < rightAnkPwmValues[ii-2]) {
    digitalWrite(49, LOW);

    transferAndWait ('t');  // add command
    transferAndWait (0);
    hb1 = transferAndWait (1);
    lb1 = transferAndWait (0);

    // disable Slave Select
    digitalWrite(49, HIGH);

    at1 = hb1;              //send x_high to rightmost 8 bits
    at1 = at1 << 8;         //shift x_high over to leftmost 8 bits
    at1 |= lb1;
    delay (100);
     rightAnkStartPosition  = at1;
   }
   analogWrite(rightAnkPwm, 0);
   digitalWrite(rightAnkEnable,LOW);
   digitalWrite(rightAnkDirection,HIGH); ////////////////////////////////////////////////////////////////////////

  }
}// End right ankle verif.
```

```
    if (driversEn[1] == 1) {// right knee verif:
      /* Serial.println(rightKneStartPosition);
       Serial.println(ii);
       Serial.println(rightKnePwmValues[ii-2]);

       while (Serial.available() == 0){
      //
      ;
    }*/
      if (rightKneStartPosition > rightKnePwmValues[ii-2]) {
        digitalWrite(rightKneDirection,HIGH); ////////////////////////////////////////////////////////////////////
        analogWrite(rightKnePwm, 50);
        digitalWrite(rightKneEnable,HIGH);
        while (rightKneStartPosition > rightKnePwmValues[ii-2]) {
          digitalWrite(49, LOW);

transferAndWait ('r');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(49, HIGH);

ar1 = hb1;              //send x_high to rightmost 8 bits
ar1 = ar1 << 8;       //shift x_high over to leftmost 8 bits
ar1 |= lb1;

delay (100);  // 1 second delay
          rightKneStartPosition  = ar1;
        }
        analogWrite(rightKnePwm, 0);
        digitalWrite(rightKneEnable,LOW);
        digitalWrite(rightKneDirection,HIGH); ////////////////////////////////////////////////////////////////////
      }
      else if (rightKneStartPosition < rightKnePwmValues[ii-2]) {
        digitalWrite(rightKneDirection,LOW); ////////////////////////////////////////////////////////////////////
        analogWrite(rightKnePwm, 50);
        digitalWrite(rightKneEnable,HIGH);
        while (rightKneStartPosition < rightKnePwmValues[ii-2]) {
```

```
        rightKneStartPosition  = ar1;
       }
       analogWrite(rightKnePwm, 0);
       digitalWrite(rightKneEnable,LOW);
       digitalWrite(rightKneDirection,HIGH); ////////////////////////////////////////////////////////////////////////
     }
   }// End right knee verif.

   if (driversEn[2] == 1) {// right hip verif:
    /* Serial.println(rightHipStartPosition);
      Serial.println(ii);
      Serial.println(rightHipPwmValues[ii-2]);

      while (Serial.available() == 0){
     //
     ;
   }*/


     if (rightHipStartPosition > rightHipPwmValues[ii-2]) {
       digitalWrite(rightHipDirection,HIGH); ////////////////////////////////////////////////////////////////////////
       analogWrite(rightHipPwm, 50);
       digitalWrite(rightHipEnable,HIGH);
       while (rightHipStartPosition > rightHipPwmValues[ii-2]) {
          digitalWrite(SS, LOW);

transferAndWait ('c');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(SS, HIGH);

ac1 = hb1;
ac1 = ac1 << 8;
ac1 |= lb1;



delay (100);
```

```
        rightHipStartPosition  = ac1;
       }
       analogWrite(rightHipPwm, 0);
       digitalWrite(rightHipEnable,LOW);
       digitalWrite(rightHipDirection,HIGH); ////////////////////////////////////////////////////////////////////
     }
     else if (rightHipStartPosition < rightHipPwmValues[ii-2]) {
       digitalWrite(rightHipDirection,LOW); ////////////////////////////////////////////////////////////////////
       analogWrite(rightHipPwm, 50);
       digitalWrite(rightHipEnable,HIGH);
       while (rightHipStartPosition < rightHipPwmValues[ii-2]) {
         digitalWrite(SS, LOW);

transferAndWait ('c');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(SS, HIGH);

ac1 = hb1;
ac1 = ac1 << 8;
ac1 |= lb1;



delay (100);
        rightHipStartPosition  = ac1;
       }
       analogWrite(rightHipPwm, 0);
       digitalWrite(rightHipEnable,LOW);
       digitalWrite(rightHipDirection,HIGH); ////////////////////////////////////////////////////////////////////
     }
   }// End right hip verif.


   ////////////////////////// End verification. //////////////////////////

   ////////////////////////// Wise verification. //////////////////////////
```

```
    for (int j=0; j < ii-2; j++){

      if (driversEn[0] == 1) {// right ankle wise:
        if (rightAnkPwmValues[j] < 0) {
          rightAnkWiseValue[j]=HIGH; ////////////////////////////////////////////////////////////////////
          rightAnkPwmValues[j]=-rightAnkPwmValues[j];
        }
        else if (rightAnkPwmValues[j] >= 0) {
          rightAnkWiseValue[j]=LOW; ////////////////////////////////////////////////////////////////////
        }
      }// End right ankle.
  ///////cambie HIGH por LOW y viceversa
      if (driversEn[1] == 1) {// right knee wise:
        if (rightKnePwmValues[j] < 0) {
          rightKneWiseValue[j]=HIGH; ////////////////////////////////////////////////////////////////////
          rightKnePwmValues[j]=-rightKnePwmValues[j];
        }
        else if (rightKnePwmValues[j] >= 0) {
          rightKneWiseValue[j]=LOW; ////////////////////////////////////////////////////////////////////
        }
      }// End right knee.

      if (driversEn[2] == 1) {// right hip wise:
        if (rightHipPwmValues[j] < 0) {
          rightHipWiseValue[j]=LOW; ////////////////////////////////////////////////////////////////////
          rightHipPwmValues[j]=-rightHipPwmValues[j];
        }
        else if (rightHipPwmValues[j] >= 0) {
          rightHipWiseValue[j]=HIGH; ////////////////////////////////////////////////////////////////////
        }
      }// End right hip.

    }
      ////////////////////////////////////////////////////////////////// Start movement:
//////////////////////////////////////////////////////////////////
Serial.println(1);
while (Serial.available() == 0){
      // Ingreso de número de repeticiones
      ;
    }
```

```
      analogWrite(rightAnkPwm, 0);
      analogWrite(rightKnePwm, 0);
      analogWrite(rightHipPwm, 0);

      digitalWrite(rightAnkEnable,HIGH);
      digitalWrite(rightKneEnable,HIGH);
      digitalWrite(rightHipEnable,HIGH);

      first=1;
      for (int i=1; i <= times; i++) {
        int h=50;
        for (int j=0; j < ii-2; j++) {
          ////////////////////////// right leg movement. //////////////////////////
          stOp == 0;
          if(h==100){
            h==0;
          }
          digitalWrite(rightAnkDirection,rightAnkWiseValue[j]);
          digitalWrite(rightKneDirection,rightKneWiseValue[j]);
          digitalWrite(rightHipDirection,rightHipWiseValue[j]);



timeZero = millis();
          timeTwo = timeZero;
           analogWrite(rightAnkPwm, 100);
          analogWrite(rightKnePwm, 100);
          analogWrite(rightHipPwm, 100);



          while (timeTwo - timeZero < inbetweenTime && stOp == 0) {
            timeTwo = millis();
          }

          analogWrite(rightAnkPwm, 0);
          analogWrite(rightKnePwm, 0);
          analogWrite(rightHipPwm, 0);

          if (first == i) {
```

```
        digitalWrite(SS, LOW);

transferAndWait ('c');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(SS, HIGH);

ac1 = hb1;
ac1 = ac1 << 8;
ac1 |= lb1;

digitalWrite(49, LOW);

transferAndWait ('r');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select

ar1 = hb1;            //send x_high to rightmost 8 bits
ar1 = ar1 << 8;       //shift x_high over to leftmost 8 bits
ar1 |= lb1;

transferAndWait ('t');  // add command
transferAndWait (0);
hb1 = transferAndWait (1);
lb1 = transferAndWait (0);

// disable Slave Select
digitalWrite(49, HIGH);

at1 = hb1;            //send x_high to rightmost 8 bits
at1 = at1 << 8;       //shift x_high over to leftmost 8 bits
at1 |= lb1;

delay (100);
```

```
        rightAnkPositionFeedback[j] = at1;
        rightKnePositionFeedback[j] = ar1;
        rightHipPositionFeedback[j] = ac1;

      }
    }
  }

  digitalWrite(rightAnkEnable,LOW);
  digitalWrite(rightKneEnable,LOW);
  digitalWrite(rightHipEnable,LOW);

  ////////////////////////// ----  //////////////////////////

  ////////////////////////////////////////////////////////////////////////// Data sending:
//////////////////////////////////////////////////////////////////////////

  ////////////////////////// right leg data sending. //////////////////////////

  if (driversEn[0] == 1) {// right ankle:
    Serial.println(rightAnkStartPosition);
    for (int j=0; j < ii-2; j++) {
      Serial.println(rightAnkPositionFeedback[j]);
    }
    Serial.println(6666);

    while (Serial.available() == 0) {
      // Confirmación.
      ;
    }

    if (Serial.available() > 0) {
      Serial.parseInt();
    }
    Serial.parseInt();
  }// End right ankle.

  if (driversEn[1] == 1) {// right knee:
    Serial.println(rightKneStartPosition);
    for (int j=0; j < ii-2; j++){
```

```
        Serial.println(rightKnePositionFeedback[j]);
      }
      Serial.println(6666);

      while (Serial.available() == 0){
        // Confirmación.
        ;
      }

      if (Serial.available() > 0) {
        Serial.parseInt();
      }
      Serial.parseInt();
    }// End right knee.

    if (driversEn[2] == 1) {// right hip:
      Serial.println(rightHipStartPosition);
      for (int j=0; j < ii-2; j++){
        Serial.println(rightHipPositionFeedback[j]);
      }
      Serial.println(6666);

      while (Serial.available() == 0){
        // Confirmación.
        ;
      }

      if (Serial.available() > 0) {
        Serial.parseInt();
      }
      Serial.parseInt();
    }// End right hip.

    ////////////////////////////////// End of data sending. //////////////////////////////////
    ///////////////// End of movement execution. /////////////////

   }
  }
  }
}
```

APÉNDICE D

Algoritmo de Python para la Raspberry Pi [Adaptado de Mármol & Sánchez, 2019]

```
import serial, time
import numpy as np
import openpyxl
import random


excel_document = openpyxl.load_workbook('mmc31.xlsx')
sheet = excel_document.get_sheet_by_name('Joint Rotations')
#Los arreglos comienzan con indice 0 mientras que en Mathlab comienza en 1
print("Recuerde revisar todas las conexiones.\n")
#com = input("Seleccione puerto COM a utilizar: ");
#com = com.upper();
arduino = serial.Serial("COM7", 9600)
time.sleep(2)
if arduino.is_open != True:
    arduino.open()
#rawString = arduino.readline();
#print(rawString);
x = np.arange(101)
x=x/100
w_Values = np.zeros((6,200))
w_Old_Values = np.zeros((6,200))
yreal = np.zeros(((len(x)-1),1))
eta = 0.4
eta_ = np.ones((6,100))
eta_ = eta_*0.4
k = 0.4
a = 0
str_Degrees = 0
str_Vels = 0
relacion = np.arange(6)*0
op = "0"
seleccionrutina = 0
zero = np.arange(3)*0
ninety = np.arange(3)*0
factor_Start_Point = np.arange(3)*0
factor = np.arange(3)*0.0
legs = ["Right"]
joints = ["Ankle","Knee","Hip"]
q3 = 0
stop3_2 = 0
conf_51 = np.arange(3)*0
conf_52 = np.arange(3)*0
conf_5 = 5101
for i in range(3):
    conf_51[i] = conf_5
    conf_52[i] = conf_5+100
    conf_5 = conf_5+10
limits_Done = 0
hi_Limit = np.arange(3)*0
lo_Limit = np.arange(3)*0
en_Pos = np.arange(3)*0
####################
```

```
conf = 0;
art="";
op2_1=0;
times =""
####################
p = input("¿Cargar Valores Iniciales? ")
if p == "1":
    t=0
    for i in legs:
        for j in joints:
            sheetName=i+" "+j+" Data"
            print(sheetName)
            zero[t]=int(openpyxl.load_workbook('Exoskeleton
Database.xlsx').get_sheet_by_name(sheetName)['C2'].value)
            ninety[t]=int(openpyxl.load_workbook('Exoskeleton
Database.xlsx').get_sheet_by_name(sheetName)['C3'].value)
            t=t+1
tom_secure = 0
tom=input("¿Desea habilitar motores? ")
if tom == "1":
    print("HABILITACION DE MOTORES")
    #print("1: Habilitar. ")
    #print("0: No Habilitar. ")
    #drivers_En=input("(ra,rk,rh): ")
    drivers_En="1,1,1"
    drivers_En_Int=list(map(int, drivers_En.split(",")))
    p_me=1
    for i in range(3):
        if drivers_En_Int[i] !=1 and drivers_En_Int[i] !=0:
            p_me=0
    while len(drivers_En_Int)!=3 or p_me==0:
        print("Ingrese valores apropiadamente.")
        drivers_En = input("(ra,rk,rh): ")
        drivers_En_Int = list(map(int, drivers_En.split(",")))
        for i in range(3):
            if drivers_En_Int[i] != 1 and drivers_En_Int[i] != 0:
                p_me = 1
    arts = []
    if drivers_En_Int[0]==1:
        arts.append("Right ankle")
    if drivers_En_Int[1]==1:
        arts.append("Right knee")
    if drivers_En_Int[2]==1:
        arts.append("Right hip")
    arduino.write(b'11111')
    while arduino.inWaiting() == 0:
        1
    if arduino.readline().decode("utf-8") == "11\r\n":
        arduino.write(drivers_En.encode())
        print("Recibimiento de datos listo.")
    while arduino.inWaiting() == 0:
        1
    if arduino.readline().decode("utf-8") == "12\r\n":
        print("Motores Listos")
    if sum(drivers_En_Int) == 0:
        tom_secure = 0
    else:
        tom_secure = 1
while op!="6":
```

```
print("MENU PRINCIPAL")
print("0. Posicion Inicial.")
print("1. Habilitar/Deshabilitar motores.")
print("2. Movimiento Libre.")
print("3. Determinar Relacion Velocidad/rpm")
print("4. Seleccionar Rutina")
print("5. Ejecutar Rutina")
print("6. Finalizar Programa")
op = input("Elija una Opcion: ")
if op=="0":
    arduino.write(b"22200")
    while arduino.inWaiting() == 0:
        1
    arduino.readline().decode("utf-8")
if op =="1": #Habilitar/Deshabilitar motores
    print("Habilitacion de Motores")
    print("1: Habilitar.")
    print("0: No habilitar.")
    drivers_En = input("(ra,rk,rh): ")
    drivers_En_Int = list(map(int, drivers_En.split(",")))
    p_me = 1
    for i in range(3):
        if drivers_En_Int[i] != 1 and drivers_En_Int[i] != 0:
            p_me = 0
    while len(drivers_En_Int) != 3 | p_me == 0:
        print("Ingrese valores apropiadamente.")
        drivers_En = input("(ra,rk,rh): ")
        drivers_En_Int = list(map(int, drivers_En.split(",")))
        for i in range(3):
            if drivers_En_Int[i] != 1 and drivers_En_Int[i] != 0:
                p_me = 1
    arts = []
    if drivers_En_Int[0] == 1:
        arts.append("Right ankle")
    if drivers_En_Int[1] == 1:
        arts.append("Right knee")
    if drivers_En_Int[2] == 1:
        arts.append("Right hip")
    arduino.write(b'11111')
    while arduino.inWaiting() == 0:
        1
    if arduino.readline().decode("utf-8") == "11\r\n":
        arduino.write(drivers_En.encode())
        print("Recibimiento de datos listo.")
    while arduino.inWaiting() == 0:
        1
    if arduino.readline().decode("utf-8") == "12\r\n":
        print("Motores Listos")
    if sum(drivers_En_Int) == 0:
        tom_secure = 0
    else:
        tom_secure = 1
while op=="2" and tom_secure==1:
    print("Movimiento Libre")
    print("Seleccione Articulacion: ")
    art_fm = input("ra,rk,rh: ").lower()
    if art_fm == "ra":
        art = "11050"
        art_str = "Right ankle"
```

```
            art_str2 = "Right Ankle"
            op2_1 = 1
            conf = 201
        elif art_fm == "rk":
            art = "11051"
            art_str = "Right knee"
            art_str2 = "Right Knee"
            op2_1 = 1
            conf = 211
        elif art_fm == "rh":
            art = "11052"
            art_str = "Right hip"
            art_str2 = "Right Hip"
            op2_1 = 1
            conf = 221
        elif art_fm == "0":
            break
        arduino.write(b"11000")
        while arduino.inWaiting() == 0:
            1
        if arduino.readline().decode("utf-8") == op+"\r\n":
            arduino.write(art.encode())
            print("Articulacion "+art_str+" Seleccionada")
        while arduino.inWaiting() == 0:
            1
        if arduino.readline().decode("utf-8") == str(conf)+"\r\n":
            conf = conf+1
            while op2_1 == 1:
                i=0
                b2 = 0
                pwm_Val = input("Valor(es) PWM(-255,255): ")
                pwm_Eval = list(map(float, pwm_Val.split(",")))
                pwm_Length = len(pwm_Eval)
                while i < pwm_Length:
                    if pwm_Eval[i]<-255 or pwm_Eval[i]>255 or
pwm_Eval[i]!=round(pwm_Eval[i]):
                        print("Seleccione un valor(es) dentro del rango")
                        pwm_Val = input("Valor(es) PWM(-255,255): ")
                        pwm_Eval = list(map(float, pwm_Val.split(",")))
                        pwm_Length = len(pwm_Eval)
                        b2 = 1
                    i = i+1
                    if b2==1:
                        i = 0
                        b2 = 0
                pwm_Val = str(pwm_Eval[0])
                for i in range(1,pwm_Length):
                    pwm_Val=pwm_Val+","+str(pwm_Eval[i])
                arduino.write(pwm_Val.encode())
                while arduino.inWaiting() == 0:
                    1
                if arduino.readline().decode("utf-8") == str(conf)+"\r\n":
                    conf = conf-1
                    inbetween_Time = input("Ingrese tiempo en segundos: ")
                    inbetween_Time = str(int(inbetween_Time) * 1000)
                    arduino.write(inbetween_Time.encode())
                pwm_FeedBack = np.arange(pwm_Length+1)*0
                for i in range(pwm_Length+1):
                    pwm_FeedBack[i] =float(arduino.readline().decode("utf-8"))
```

```
                print("Posiciones Registradas")
                for i in range(pwm_Length+1):
                    print(pwm_FeedBack[i])
                op2_1_ver = input("¿Desea continuar con movimiento libre en
"+art_str+"? ")
                if op2_1_ver == "0":
                    break
                arduino.write(b"11000")
                while arduino.inWaiting() == 0:
                    1
                if arduino.readline().decode("utf-8") == op+"\r\n":
                    conf = conf+1
                    arduino.write(art.encode())
                    print("Articulacion "+art_str+" Seleccionada")
                while arduino.inWaiting() == 0:
                    1
                arduino.readline()
    while op=="3" and tom_secure==1: #Determinar relacion Velocidad/rpm
        arduino.write(b"11100")
        while arduino.inWaiting() == 0:
            1
        if arduino.readline().decode("utf-8") == op+"\r\n":
            print("DETERMINACION FACTOR VELOCIDAD/RPM")
            print("Seleccione Articulacion: ")
            art_vf = input("ra,rk,rh: ").lower()
            if art_vf == "ra":
                art = "11150"
                art_str = "Right ankle"
                c = 0
                conf_3 = 301
            elif art_vf == "rk":
                art = "11151"
                art_str = "Right knee"
                c = 1
                conf_3 = 311
            elif art_vf == "rh":
                art = "11152"
                art_str = "Right hip"
                c = 2
                conf_3 = 321
            elif art_vf == "0":
                arduino.write(b"111000")
                break
            if p != 1:
                o = "0"
                while o !="1":
                    t=0
                    for i in legs:
                        for j in joints:
                            sheetName = i + " " + j + " Data"
                            zero[t] = int(openpyxl.load_workbook('Exoskeleton
Database.xlsx').get_sheet_by_name(sheetName)['C2'].value)
                            ninety[t] = int(openpyxl.load_workbook('Exoskeleton
Database.xlsx').get_sheet_by_name(sheetName)['C3'].value)
                            factor_Start_Point[t] =
int(openpyxl.load_workbook('Exoskeleton
Database.xlsx').get_sheet_by_name(sheetName)['F2'].value)
                            t = t + 1
                    print(" referencial angular positions.\n0°:
```

```
"+str(zero[c])+"\n90°: "+str(ninety[c])+"\n")
                    o=input("¿Está de acuerdo? ")
                    if o!="1":
                        q3 = input("¿Desea regresar al menú principal? ")
                        o="1"
            else:
                o="0"
                while o!="1":
                    print(" referencial angular positions.\n0°:
"+str(zero[c])+"\n90°: "+str(ninety[c])+"\n")
                    o = input("¿Está de acuerdo? ")
                    while o!="1":
                        t=0
                        for i in legs:
                            for j in joints:
                                sheetName = i + " " + j + " Data"
                                zero[t] = int(openpyxl.load_workbook('Exoskeleton
Database.xlsx').get_sheet_by_name(sheetName)['C2'].value)
                                ninety[t] = int(openpyxl.load_workbook('Exoskeleton
Database.xlsx').get_sheet_by_name(sheetName)['C3'].value)
                                factor_Start_Point[t] =
int(openpyxl.load_workbook('Exoskeleton
Database.xlsx').get_sheet_by_name(sheetName)['F2'].value)
                                t = t + 1
                                print(" referencial angular positions.\n0°:
"+str(zero[c])+"\n90°: "+str(ninety[c])+"\n")
                                o = input("¿Está de acuerdo? ")
                                if o != "1":
                                    q3 = input("¿Desea regresar al menú principal? ")
                                    o = "1"
                    t=0
                    for i in legs:
                        for j in joints:
                            sheetName = i + " " + j + " Data"
                            factor_Start_Point[t] =
int(openpyxl.load_workbook('Exoskeleton
Database.xlsx').get_sheet_by_name(sheetName)['F2'].value)
                            t = t + 1
            if q3 == "1":
                arduino.write(b"111000")
                q3="0"
                break
            arduino.write(art.encode())
            while arduino.inWaiting() == 0:
                1
            if arduino.readline().decode("utf-8") == str(conf_3)+"\r\n":
                print("posicion inicial: "+str(factor_Start_Point[c]))
                input("¡Atención: No obstruir el campo de movimiento del
exoesqueleto!")
                arduino.write(str(factor_Start_Point[c]).encode())
            while arduino.inWaiting() == 0:
                1
            firstvalue = float(arduino.readline().decode("utf-8"))
            secondvalue = float(arduino.readline().decode("utf-8"))
            omega = abs((90/(ninety[c]-zero[c]))*(secondvalue-firstvalue))/1.5
            factor[c] = (100-33)/omega
            print("Factor PWM : "+str(factor[c]))
            relacion[c]=1
    if op == "4" and tom_secure == 1: #Seleccionar Rutina
```

```
#filename = "mmc3"
#sheet = "Joint Rotations";np.zeros((3, 1))
hip_Des_Vel = np.arange(100)*0.0
knee_Des_Vel = np.arange(100)*0.0
ankle_Des_Vel = np.arange(100)*0.0
pwm_Values = np.zeros((3, 100))
pwm_Values_Real = np.zeros((3, 100))
art_Start_Ang = np.arange(3)*0
art_Vel = np.zeros((3, 100))
art_Vel_Las = np.zeros((3, 100))
art_Des_Vel = np.zeros((3, 100))
art_Vel_Error =np.arange(3)*0
q = 0
print(' (N) Marcha a rapidez natural.')
print('(XS) Marcha muy lenta.')
print(' (S) Marcha lenta.')
print(' (M) Marcha a rapidez media.')
print(' (L) Marcha Rápida.')
print(' (T) Marcha en puntillas.')
print(' (H) Marcha en talones.')
print(' (U) Marcha subiendo escaleras.')
print(' (D) Marcha subiendo escaleras.')
rut_cs = input('Seleccione rutina: ')
if rut_cs == "N":
    row = 'D'
elif rut_cs =="XS":
    row = 'G'
elif rut_cs == "S":
    row = 'J'
elif rut_cs == "M":
    row = 'M'
elif rut_cs == "L":
    row = 'P'
elif rut_cs == "T":
    row = 'S'
elif rut_cs == "H":
    row = 'V'
elif rut_cs == "U":
    row = 'Y'
elif rut_cs == "D":
    row = 'AB'
delay = str((round(int(input('Ingrese duración de ciclo: '))/100)*1000)
if drivers_En_Int[0] == 1:
    i=0
    for row in range(710, 810):
        r = sheet.cell(row = (row+1), column = 4).value
        t = sheet.cell(row=row, column=4).value
        ankle_Des_Vel[i] = (r - t)/(float(delay)/1000);
        i=i+1
if drivers_En_Int[1] == 1:
    i = 0
    for row in range(609, 709):
        r = sheet.cell(row=(row + 1), column=4).value
        t = sheet.cell(row=row, column=4).value
        knee_Des_Vel[i] = (r - t) / (float(delay) / 1000);
        i = i + 1
if drivers_En_Int[2] == 1:
    i = 0
    for row in range(306, 406):
```

```
                r = sheet.cell(row=(row + 1), column=4).value
                t = sheet.cell(row=row, column=4).value
                hip_Des_Vel[i] = (r - t) / (float(delay) / 1000);
                i = i + 1
        for i in range(3):
            if drivers_En_Int[i] == 1:
                if i==0:
                    art_Start_Ang[i] = round(zero[i] + ((ninety[i]-
zero[i])/90)*(sheet.cell(row=710, column=4).value))
                    for j in range(100):
                        pwm_Values[i][j] = factor[i]*ankle_Des_Vel[j]
                    for t in range((len(x)-1)*2):
                        w_Values[i][t] = random.uniform(0, 1)
                    for j in range(100):
                        art_Des_Vel[i][j]=ankle_Des_Vel[j]
                    en_Pos[q] = i
                    q = q+1
                if i==1:
                    art_Start_Ang[i] = round(zero[i] + ((ninety[i] - zero[i]) / 90)
*(sheet.cell(row=609, column=4).value))
                    for j in range(100):
                        pwm_Values[i][j] = factor[i] * knee_Des_Vel[j]
                    for t in range((len(x) - 1) * 2):
                        w_Values[i][t] = random.uniform(0, 1)
                    for j in range(100):
                        art_Des_Vel[i][j] = knee_Des_Vel[j]
                    en_Pos[q] = i
                    q = q + 1
                if i==2:
                    art_Start_Ang[i] = round(zero[i] + ((ninety[i] - zero[i]) / 90) *
(sheet.cell(row=306, column=4).value))
                    for j in range(100):
                        pwm_Values[i][j] = factor[i] * hip_Des_Vel[j]
                    for t in range((len(x) - 1) * 2):
                        w_Values[i][t] = random.uniform(0, 1)
                    for j in range(100):
                        art_Des_Vel[i][j] = hip_Des_Vel[j]
                    en_Pos[q] = i
                    q = q + 1
        b = np.arange(3)*0
        l = np.arange(3)*0
        while sum(b)==0:
            art_Vel_Las_Las = art_Vel_Las
            art_Vel_Las = art_Vel
            for i in range(3):
                if drivers_En_Int[i]==1:
                    t=0
                    for j in range(100):
                        art_Vel[i][j]=1.06*w_Values[i][t]+1.06*w_Values[i][t+1]
                        t=t+2
            for i in range(len(en_Pos)):
                art_Vel_Error[en_Pos[i]]=0
            for i in range(len(en_Pos)):
                for j in range(100):
                    art_Vel_Error[en_Pos[i]] =
art_Vel_Error[en_Pos[i]]+abs(art_Vel[en_Pos[i],j]-art_Des_Vel[en_Pos[i],j])
            for i in range(len(en_Pos)):
                if art_Vel_Error[en_Pos[i]] < 0.1:
                    b[i]=1
```

```
                else:
                    j=0
                    l[i]=l[i]+1
                    for t in range(len(x)-1):
                        if l[en_Pos[i]]>2:
                            if (art_Vel[en_Pos[i],t]-
art_Des_Vel[en_Pos[i],t])*(art_Vel_Las[en_Pos[i],t]-art_Des_Vel[en_Pos[i],t])<0:
                                eta_[en_Pos[i], t] = 0.6 * eta_[en_Pos[i], t]
                            elif (art_Vel[en_Pos[i],t]-
art_Des_Vel[en_Pos[i],t])*(art_Vel_Las[en_Pos[i],t]-art_Des_Vel[en_Pos[i],t])>0 and
(art_Vel_Las[en_Pos[i],t]-art_Des_Vel[en_Pos[i],t])*(art_Vel_Las_Las[en_Pos[i],t]-
art_Des_Vel[en_Pos[i],t])>=0:
                                eta_[en_Pos[i], t] = 1.07 * eta_[en_Pos[i], t]
                        if art_Vel[en_Pos[i],t]!=art_Des_Vel[en_Pos[i],t]:
                            w_Values[en_Pos[i], j] = w_Values[en_Pos[i], j] -
eta_[en_Pos[i], t]*(art_Vel[en_Pos[i], t]-art_Des_Vel[en_Pos[i],
t])*1.06+k*(w_Values[en_Pos[i], j] - w_Old_Values[en_Pos[i], j])
                            w_Values[en_Pos[i], j + 1] =  w_Values[en_Pos[i], j+1] -
eta_[en_Pos[i], t]*(art_Vel[en_Pos[i], t]-art_Des_Vel[en_Pos[i],
t])*1.06+k*(w_Values[en_Pos[i], j+1] - w_Old_Values[en_Pos[i], j+1])
                            w_Old_Values[en_Pos[i], j] = w_Values[en_Pos[i], j]
                            w_Old_Values[en_Pos[i], j + 1] = w_Values[en_Pos[i], j +
1]
                        j = j+2
                    b[i]=0
            print("Numero de Iteraciones")
            seleccionrutina = 1
        if op=="5" and tom_secure==1: #Ejecutar Rutina
            if sum(relacion)>=0:
                if seleccionrutina==1:
                    t=0
                    if limits_Done == 0:
                        for i in legs:
                            for j in joints:
                                sheetName = i + " " + j + " Data"
                                lo_Limit[t] = int(openpyxl.load_workbook('Exoskeleton
Database.xlsx').get_sheet_by_name(sheetName)['M2'].value)
                                hi_Limit[t] = int(openpyxl.load_workbook('Exoskeleton
Database.xlsx').get_sheet_by_name(sheetName)['M3'].value)
                                t = t + 1
                    op3="5"
                    while op3!="3":
                        print('1. Prueba 1.')
                        print('2. Ejecutar rutina.')
                        print('3. Regresar.')
                        op3 = input('Elija una opción:')
                        if op3 == "1":
                            arduino.write(b"11200")
                            while arduino.inWaiting() == 0:
                                1
                            arduino.readline().decode("utf-8")
                        while op3=="2":
                            if stop3_2==1:
                                stop3_2=0
                                break
                            arduino.write(b"11300")
                            #CReacion de csv
                            for i in range(len(en_Pos)):
                                t=0
```

```
                            for j in range(100):
                                art_Vel[en_Pos[i], j] = 1.06 * w_Values[en_Pos[i], t]
+ 1.06 * w_Values[en_Pos[i],t + 1]
                                t = t + 2
                        art_Mov = ""
                        for i in range(len(en_Pos)):
                            for j in range(100):
                                pwm_Values[en_Pos[i], j] = factor[en_Pos[i]] *
art_Vel[en_Pos[i], j]
                                if j==0:
                                    if pwm_Values[en_Pos[i],1]<0:
                                        art_Mov =
art_Mov+str(int(round(pwm_Values[en_Pos[i],1]-35)))
                                    else:
                                        art_Mov =
art_Mov+str(int(round(pwm_Values[en_Pos[i],1]+35)))
                                else:
                                    if pwm_Values[en_Pos[i], j] < 0:
                                        art_Mov = art_Mov+
","+str(int(round(pwm_Values[en_Pos[i], j] - 35)))
                                    else:
                                        art_Mov = art_Mov +
","+str(int(round(pwm_Values[en_Pos[i], j] + 35)))
                            art_Mov=art_Mov+","+str(art_Start_Ang[en_Pos[i]])+"/"
                        art_Mov = art_Mov.split("/")
                        for j in range(101):
                            if j<=51:
                                1
                                #ankle_Data2[j] = ankle_Data[j + 50]
                                # knee_Data2[j] = knee_Data[j + 50]
                                # hip_Data2[j] = hip_Data[j + 50]
                            else:
                                1
                                #ankle_Data2[j] = ankle_Data[j - 50]
                                # knee_Data2[j] = knee_Data[j - 50]
                                # hip_Data2[j]= hip_Data[j - 50]
                        for j in range(100):
                            if j <= 50:
                                1
                                #ankle_vel2[j] = ankle_Des_Vel[j + 50]
                                # knee_vel2[j] = knee_Des_Vel[j + 50]
                                # hip_vel2[j] = hip_Des_Vel[j + 50]
                            else:
                                1
                                #ankle_vel2[j] = ankle_Des_Vel[j - 50]
                                # knee_vel2[j] = knee_Des_Vel[j - 50]
                                # hip_vel2[j] = hip_Des_Vel[j - 50]
                        while arduino.inWaiting() == 0:
                            1
                        if arduino.readline().decode("utf-8") == "52\r\n":
                            for i in range(len(en_Pos)):
                                print(art_Mov[i])
                                arduino.write(art_Mov[i].encode())
                                while arduino.inWaiting() ==0:
                                    1
                                if arduino.readline().decode("utf-8") ==
str(conf_52[en_Pos[i]])+"\r\n":
                                    print("data Sent")
                                    go = input("data Sent")
```

```
                            else:
                                while True:
                                    stop = input("data not sent. STOP PROGRAM")
                        arduino.write(delay.encode())
                        while arduino.inWaiting() ==0:
                            1
                        if arduino.readline().decode("utf-8") == "5200\r\n":
                            times= input("¿Cuántas veces desea repetir el ciclo? ")
                        arduino.write(times.encode())


######################################################################


                        while arduino.inWaiting() == 0:
                            1
                        print("Comienzo de Movimiento")
                        print(arduino.readline().decode("utf-8"))
                        input("continuar...")
                        arduino.write(b'1')

                        while arduino.inWaiting() == 0:
                            1
                        out_Art2 = np.zeros((3,101))
                        for i2 in range(len(en_Pos)):
                            for i in range(101):
                                out_Art2[en_Pos[i2],i] =
arduino.readline().decode("utf-8")
                            if arduino.readline().decode("utf-8")=="6666\r\n":
                                arduino.write(b"1")
                                print("Finished")
                        stop3_2=1
            else:
                print("No ha seleccionado una rutina.")
        else:
            print("No ha sido determinada la relacion.")

arduino.close()
```