CREACIÓN DE LIBRERÍA EN C/C++ PARA EL CONTROL DE PRÓTESIS DE MIEMBRO SUPERIOR BASADA EN ACTUADORES DC DESTINADA A LA FUNDACIÓN MATERALIZACIÓN 3D

Juan Andrés González Urquijo

Práctica Profesional

Tutor
M.Sc. Pedro Antonio Aya Parra
Johan García

UNIVERSIDAD DEL ROSARIO
ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO
PROGRAMA DE INGENIERÍA BIOMÉDICA
BOGOTÁ D.C
2021

AGRADECIMIENTOS

Agradezco a mi familia y amigos, quienes me acompañaron y apoyaron a lo largo de mis estudios, dándome la fuerza y el equilibrio necesario para dar lo mejor de mí. A mis maestros del programa de Ingeniería Biomédica, quienes indudablemente, fueron un pilar para mi desarrollo profesional y en especial al Ingeniero Juan Manuel López por ser un gran mentor.

Especial agradecimiento al Ingeniero Pedro Antonio Haya y a Johan García, por su apoyo y comprensión durante el desarrollo de mis prácticas, ya que fueron de vital importancia para mi desarrollo como Ingeniero Biomédico.

Agradezco a la Universidad del Rosario, por su interés constante en mi bienestar como estudiante y como futuro profesional, a la Universidad Escuela Colombia de Ingeniería Julio Garavito, por su aporte indispensable para mi formación, y al Fondo de Solidaridad de la ECI por ayudarme en los momentos más difíciles de la carrera.

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

LISTA DE ANEXOS.

1. INTRODUCCIÓN

Actualmente en el mercado existen diversas prótesis de miembro superior, sin embargo la accesibilidad es muy reducida, ya que pueden llegar a costar entre 4,000 y 10,000 USD, si se basan en la energía corporal, y entre 25,000 y 75,000 USD si son basadas en fuentes de energía externa [1][2]. Una de las alternativas es el uso de impresoras 3D para la construcción de las prótesis de miembro superior, y existe evidencia que indica una mejora en la calidad de vida del usuario final [3], de todos modos, es necesario seguir investigando y mejorando los diseños actuales [4].

Teniendo en cuenta lo mencionado, la fundación Materialización 3D (M3D), fundada por Philippe Parmentier en el 2014 en Madrid, Cundinamarca, surge como una solución para las personas que quieren acceder a una prótesis de forma gratuita. Uno de los planes de desarrollo de prótesis es “Hazlo tú mismo”, donde el beneficiario está implicado transversalmente en el desarrollo de su propia prótesis. Actualmente, la fundación se encuentra en el continuo desarrollo de prototipos y modelos para ayudar a la mayor cantidad de personas posibles dadas sus circunstancias personales.

Uno de los modelos más ambiciosos que se están desarrollando actualmente, es el desarrollo de una prótesis biónica de miembro superior con varios actuadores. Esto implica varios retos en el proceso de diseño, desde la parte mecánica hasta la electrónica y la programación. Ya que el motivo de la fundación es ayudar a la mayor cantidad de personas posibles, se busca que el modelo sea de bajo costo, por lo tanto, se usa en su mayoría herramientas de acceso libre.

Para poder manejar la prótesis correctamente, es necesario tener un algoritmo funcional y dinámico, que permita el manejo de diferentes motores de forma precisa, esto para poder otorgarle la libertad al usuario de realizar varios movimientos. Teniendo en cuenta que cada usuario posee diferencias en su modelo de prótesis (dimensiones, capacidad de carga, materiales, peso, entre otros), es necesario tener un modelo de algoritmo que sea capaz de ajustarse a las necesidades personales de cada usuario.

Para poder cumplir con este propósito, se usan comúnmente sistemas de lazo cerrado con controles tipo PID. Este modelo permite manejar diversos actuadores, como los típicos motores DC, además, este algoritmo tiene que ir integrado correctamente con el sistema físico, que corresponde a actuadores de diferente tipo, drivers, sensores de posición, limites mecánicos, fuente de alimentación, microcontrolador, entre otros.

En la fundación M3D tiene unos sistemas de preferencia ya establecidos. Los motores se eligen principalmente por sus propiedades mecánicas (como por ejemplo el material de los engranajes). Dicho esto, y por la rentabilidad tanto económica y de potencia eléctrica, es necesario usar controladores externos. Algunos motores incorporan sensores de posición integrados, que son de utilidad para el manejo de estos. En caso de no tenerlos es necesario hacer adaptaciones al diseño para poder leer esta posición. Con estos componentes mencionados, ya se puede iniciar el diseño de un control PID para manejar los motores correctamente y lograr las posiciones deseadas.

2. OBJETIVOS

2.1. Generales

1. Desarrollar una librería en C/C++ para el manejo electrónico de prótesis de miembro superior, teniendo en cuenta las condiciones y preferencias de trabajo de la fundación M3D, esto para poder facilitar y optimizar la programación de cualquier modelo de prótesis

2.2. Específicos

1. Crear la librería compatible con las tarjetas de Arduino y la ESP32
2. Simular el funcionamiento de la librería en Proteus
3. Evaluar el funcionamiento de las librerías en modelos reales dentro de un ambiente experimental
4. Escribir documentación completa del desarrollo

3. METODOLOGÍA

3.1. Problema a solucionar

Explorando la naturaleza de trabajo de la fundación M3D, cada vez que llega un nuevo practicante o voluntario, debe construir nuevamente todo el modelo de una prótesis desde su inicio, por lo tanto, no hay un procedimiento estándar que permita optimizar el desarrollo de estas. Sin embargo, en el proceso de diseño, hay unos pasos que siempre se repiten y pueden ser estandarizados, como lo es la programación de la prótesis.

Programar el movimiento teniendo en cuenta los sensores que se vayan a usar y los motores, puede ser un procedimiento largo y no siempre fácil de lograr satisfactoriamente. Dicho esto, se generó una librería que facilita el proceso de programación de la prótesis para los nuevos voluntarios u desarrolladores que lleguen a generar más modelos de prótesis, ofreciendo la oportunidad para mejorar otros aspectos de la prótesis.

Para desarrollar esta propuesta, los componentes a usar son los que la fundación M3D ha venido trabajando, por lo tanto, no implicara gastos adicionales o adaptaciones forzosas al procedimiento ya establecido. Después de una breve investigación, se determinan los sistemas tanto de software como de hardware con los que se van a trabajar.

Para el software, usan principalmente el IDE de Arduino y las librerías de libre acceso que son compatibles con este sistema. Por lo tanto, las librerías desarrolladas deben cumplir con este requisito y ser adaptables a las diversas tarjetas de Arduino que se puedan llegar a usar (Uno, nano, mega, entre otros). Respecto al Hardware y diseños electrónicos, usan por lo general tarjetas Arduino o módulos ESP32 (por su accesibilidad, funcionalidad y bajo costo), servomotores de engranajes metálicos con posibilidad de modificación, y drivers LM293X.

Con esta información, se procede a generar librerías que sean fáciles de usar para poder programar en pocas líneas de código las prótesis, con el objetivo de tener un sistema que permita experimentar con los modelos y pueda ser implementado en su totalidad. Previo a la implementación, se generarán las correspondientes simulaciones. Este código contará con adaptabilidad para diferentes modelos, motores y diseños, y se pretende probar en al menos dos montajes la funcionalidad del desarrollo.

3.2. Fases del proyecto

Este proyecto se segmentó en cinco fases de desarrollo, dichas fases permitieron la culminación del proyecto.

3.2.1. Fase 1 – Delimitación del proyecto

Para esta primera fase se buscaron áreas de mejora en las cuales se pudiera aportar algo valioso para la fundación. Para hacer esto correctamente, pasaron tres semanas de observación y diálogo con el equipo de trabajo para poder generar un proyecto teniendo en cuenta mis habilidades y las necesidades de la fundación. Posterior a esto, se generó una propuesta y se discutió con el encargado de la fundación M3D acerca de la idea del proyecto. Teniendo en cuenta sus comentarios, se generó una propuesta final.

Es importante mencionar que la mayoría de los practicantes y desarrolladores en la fundación M3D tienen fuertes conocimiento en diseño de prótesis, por lo tanto, una dificultad recurrente identificada era el diseño electrónico y/o la programación para sistemas complejos. Dicho esto, se decidió trabajar en un proyecto que uniese estos dos aspectos.

Finalmente, la propuesta considerada es generar una librería que permitiese la fácil programación de la prótesis independiente de la cantidad de motores que se vayan a usar. Además, se pretende que tenga espacio para mejoras.

3.2.2. Fase 2 – Recopilación de información

Con una propuesta inicial ya creada, se procedió a recopilar la información necesaria para poder delimitar correctamente la propuesta final. Para cumplir esto, se procedió a dialogar con los actuales trabajadores de la Fundación y la documentación de los trabajadores previos para determinar cuáles eran las principales necesidades y las condiciones en las cuales se trabajaba en cuánto al software y hardware

Posterior a esta recopilación de información, se creó un listado de elementos necesarios sobre los cuáles se va a trabajar y el desarrollo estaría destinada a funcionar. Para esta determinación, también se tuvieron en cuenta otros factores aparte de los ya mencionados como lo son los costos, la disponibilidad y la facilidad de uso

3.2.3. Fase 3 – Desarrollo

En esta tercera fase se inició con la programación de la librería para manejar la prótesis. A medida que ésta se fue desarrollando se fue probando sobre diferentes

simulaciones de un sistema similar a los usados en las prótesis en la Fundación para asegurar desde el punto de vista teórico su buen funcionamiento. Esta fase fue la que requirió más esfuerzo y tiempo, puesto que tenía que ser pensado desde diferentes puntos de vista. Los puntos para considerar fueron los siguientes:

1. Facilidad de uso: Este factor es crucial para poder completar con el objetivo inicial del proyecto, ya que si se implementaba un sistema difícil de implementar con una complejidad elevada que no permitiese manejar la librería no tendría sentido el proyecto en curso.
2. Adaptabilidad: Teniendo en cuenta que hay diferentes diseños de prótesis, es de vital importancia que el código sea adaptable a diferentes condiciones sin salirse de los parámetros mostrados en la Tabla 1. Permitiendo usar varios motores simultáneamente, diferentes parámetros de control, diferentes sensores de posición, límites de movimiento, entre otros.
3. Funcionalidad: este último punto es el más importante, puesto que, si no se tiene una funcionalidad con una precisión aceptable, todo el proyecto habrá sido en vano.

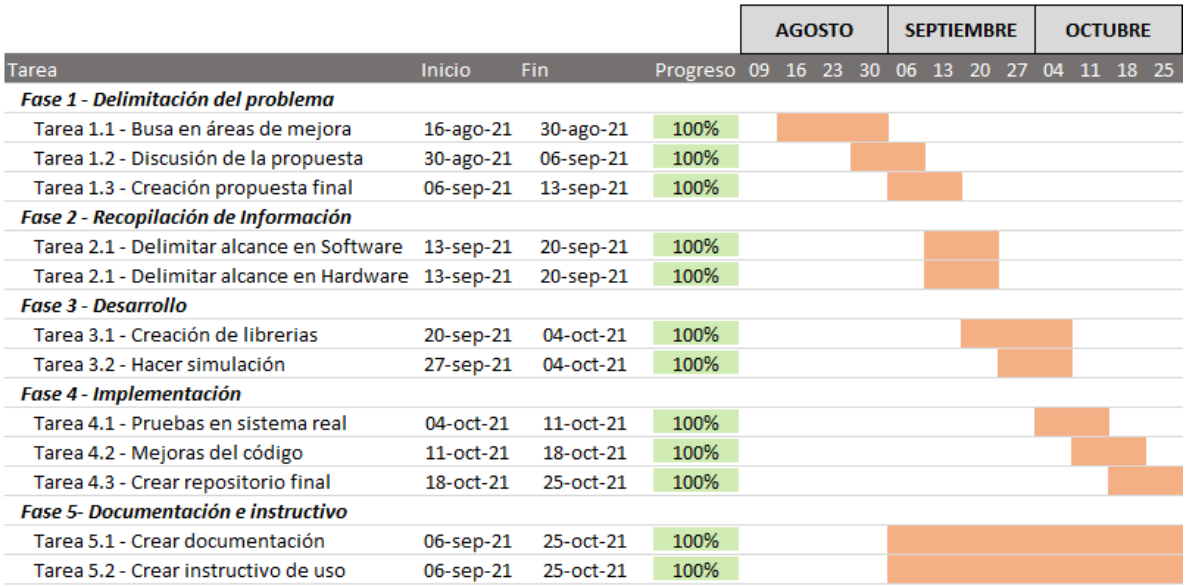
3.2.4. Fase 4 – Implementación

Después de probar el correcto funcionamiento de las librerías en simulación (Proteus), se procedió a hacer pruebas en sistemas reales. Esta fase de prueba del código permitió crear mejoras en la librería, para poder crear un repositorio final que cumpla con los requisitos descritos en la fase 3 y en general con la expectativa del diseñador.

3.2.5. Fase 5 – Documentación e instructivo

Después de ya tener el desarrollo creado es necesario crear una documentación e instructivos de uso que permita a cualquier persona implementar la librería desarrollada. esta documentación incluye una descripción de todas las funciones creadas, dónde se define su funcionalidad, las entradas, las salidas y algunos ejemplos de uso. además de esto se incluye un instructivo paso a paso para montar un ejemplo de funcionamiento.

3.3. Diagrama de Gantt



Tarea	Inicio	Fin	Progreso
Fase 1 - Delimitación del problema			
Tarea 1.1 - Busa en áreas de mejora	16-ago-21	30-ago-21	100%
Tarea 1.2 - Discusión de la propuesta	30-ago-21	06-sep-21	100%
Tarea 1.3 - Creación propuesta final	06-sep-21	13-sep-21	100%
Fase 2 - Recopilación de Información			
Tarea 2.1 - Delimitar alcance en Software	13-sep-21	20-sep-21	100%
Tarea 2.1 - Delimitar alcance en Hardware	13-sep-21	20-sep-21	100%
Fase 3 - Desarrollo			
Tarea 3.1 - Creación de librerias	20-sep-21	04-oct-21	100%
Tarea 3.2 - Hacer simulación	27-sep-21	04-oct-21	100%
Fase 4 - Implementación			
Tarea 4.1 - Pruebas en sistema real	04-oct-21	11-oct-21	100%
Tarea 4.2 - Mejoras del código	11-oct-21	18-oct-21	100%
Tarea 4.3 - Crear repositorio final	18-oct-21	25-oct-21	100%
Fase 5- Documentación e instructivo			
Tarea 5.1 - Crear documentación	06-sep-21	25-oct-21	100%
Tarea 5.2 - Crear instructivo de uso	06-sep-21	25-oct-21	100%

Figura 3.3–1. Diagrama de Gantt por semanas. Se muestras las tareas planeadas y desarrolladas a lo largo del periodo de práctica

4. RESULTADOS

Después de la culminación de la fase 1 y 2 de la metodología, se concluye que se creará unas librerías para Arduino para experimentar y controlar las prótesis desarrolladas en la fundación M3D. Para esto es necesario tener en cuenta los siguientes componentes en Hardware y Software que se utilizarán.

Tabla 1. Componentes del proyecto

Componentes	***Función***	***Tipo***
PC compatible con IDE de Arduino	Programar microcontrolador	Hardware
Arduino UNO/NANO/MEGA	Controlador de la prótesis	
ESP32	Controlador de la prótesis	
LM293D	Driver de los motores	
SERVOMOTORES	Movimientos de la prótesis	
Potenciómetros (SP)	Sensor de posición	
Sensor de intención de movimiento (SIMV)	Leer intención de movimiento	
Proteus	Simulación del sistema	Software
Arduino IDE	Programación del IDE	

En la Tabla 1 se muestran los componentes principales con los cuales se desarrollará el proyecto final. Estos fueron seleccionados dadas las condiciones de la fundación M3D

4.1. Librería

4.1.1. Mover

Mueve 1 motor asumiendo que está conectado a un controlador L293D. Esta función mantiene el movimiento hasta que se llame nuevamente la misma función o se detenga de otra manera.

4.1.2. *leer_analogo*

Lee el puerto análogo e imprime los valores en el Serial plotter

4.1.3. filtro_promedios

Adquiere datos de un puerto análogo y hace un filtro de promedios.

4.1.4. sgn

Devuelve el signo del número de entada

4.1.5. inicializar_motor_pot

Defines los pines de entrada, salida del que van al driver del motor y del análogo del potenciómetro, que marcaría la posición deseada del motor.

4.1.6. inicializar_motor

Defines los pines de entrada, salida del que van al driver del motor.

4.1.7. control_p

Implementa un control tipo p y permite un margen de error. Esta función devuelve la velocidad (que se puede usar como argumento de la función *mover*)

4.1.8. int_error

Calcula la integral del error y devuelve el valor. Si el error es bajo, se reinicia el valor de la integral a 0.

4.1.9. control_PI

Hace un control tipo PI. Devuelve la velocidad del motor.

4.1.10. control_PID

Hace un control tipo PID. Devuelve la velocidad del motor. Esta función solo multiplica el signo del error por P, ya que dado el montaje esto se tuvo que cambiar. Sin embargo, se puede modificar manualmente, cambiando en la función

4.1.11. mover_y_controlar_posicion_LC

Mueve el motor y controla la posición en lazo cerrado usando un controlador tipo PID. Es necesario enviar en la variable de control la posición (posición 2)

4.1.12. mover_y_controlar_potenciometro_LC

Mueve el motor y controla la posición en lazo cerrado usando un controlador tipo PID. El motor sigue la curva del potenciómetro.

4.1.13. Limites

Define los límites de movilidad del motor dada la posición, para evitar posicione no deseadas del motor.

Muchas de estas funciones se llaman directamente entre ellas, para así disminuir la cantidad de código en el archivo principal. Además de esto, ninguna tiene variables predeterminadas, todas pueden ser manipuladas para su experimentación.

4.2. Propuesta de Algoritmo usando Librería

Con la intención de generar un sistema fácil e intuitivo de usar, se plantea el siguiente modelo de algoritmo para manejar la prótesis. Se verá el detalle de cada bloque más adelante en esta sección.

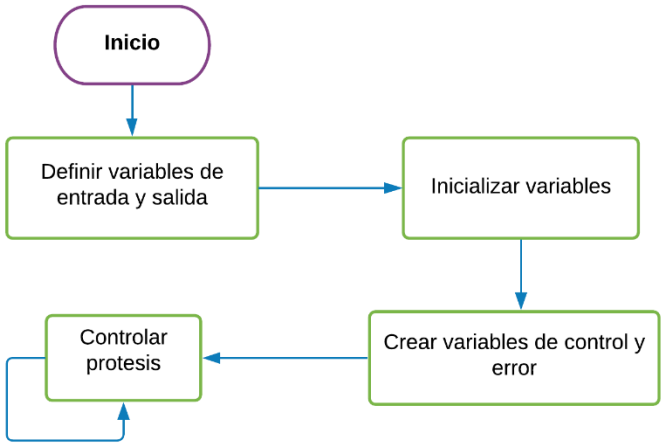


Figura 4–1. Esquema general del Algoritmo de control propuesto

La estructura propuesta en la Figura 4–1 para manejar las prótesis, permite una facilidad de poder agregar varios motores, controlar simultáneamente la posición de varios motores, generar posiciones estándar y también posiciones dependientes de la señal del sensor o señal de intención de movimiento (SIMV).

4.2.1. Definir variables de entrada y salida

Para cada uno de los motores se tienen tres salidas desde el microcontrolador y dos entradas desde el sistema, las salidas hacen referencia a las entradas del driver, dos entradas son para la dirección y una entrada es para la velocidad. Las entradas que se esperan del sistema son la posición actual del motor y la posición a la cual se quiere llevar. Esta última también puede ser definida por software si es una constante. El equivalente en código de este bloque es el siguiente:

```
// Definir variables ---------------------------------------------

//Salida al motor
const int ma[] = {2, 3, 4};
const int mb[] = {5, 6, 7};
const int mc[] = {8, 9, 10};
const int md[] = {11, 12, 13};

// Entradas
const int  sens_a = A1, pot_a = A0;
const int  sens_b = A3, pot_b = A2;
const int  sens_c = A5, pot_c = A4;
const int  sens_d = A7, pot_d = A6;
```

Figura 4–2. Código para definir entradas y salidas del motor.

En la Figura 4–2 se pueden ver una forma de definir los pines para el movimiento de cuatro motores. Hasta este punto no se ha usado ninguna de las funciones mencionadas en la sección 4.1. Para facilidad del manejo de variables, es conveniente usar arreglos de variables.

4.2.2. Inicializar Variables

En esta parte del algoritmo se definen las variables en el sistema, las funciones usadas son las descritas en la sección 4.1. Dependiendo la necesidad, se puede usar una de dos funciones, *inicializar_motor_pot* o *inicializar_motor*. En la Figura 4–3 se muestra el uso de la función en el código principal, y en la *Figura 4–4* se observa la función.

```
void setup() {

  // Open serial
  Serial.begin(9600);

  // Inicializar pines del motor
  inicializar_motor_pot(ma, sens_a, pot_a);
  inicializar_motor_pot(mb, sens_b, pot_b);
  inicializar_motor_pot(mc, sens_c, pot_c);
  inicializar_motor_pot(md, sens_d, pot_d);
}
```

Figura 4–3. Inicialización de cuatro motores en el setup.

```
void inicializar_motor_pot(int * motor, int sensor, int potenciometro) {

  // Definir pins de salida
  pinMode(motor[0], OUTPUT);
  pinMode(motor[1], OUTPUT);
  pinMode(motor[2], OUTPUT);

  // Definir pines de entrada
  pinMode(potenciometro, INPUT);
  pinMode(sensor, INPUT);

}
```

Figura 4–4. Función de la librería

4.2.3. Crear variables de control y error

Esta creación de variables está destinada a guardar un espacio en memoria y guardar organizadamente las variables usadas en todo el algoritmo de control. Se tiene un arreglo de trece variables tipo entero y una variable tipo *float* para el error. Esto es para cada motor (Figura 4–5).

```
// Variables de control  ----------------------------------------

int vars_control_a[13] = {0, 0, 0, 0, 0, 0, 200, 0.0001, 0.0001, 10, 5, sens_a, pot_a};
int vars_control_b[13] = {0, 0, 0, 0, 0, 0, 250, 0.0001, 0.0001, 10, 5, sens_b, pot_b};
int vars_control_c[13] = {0, 0, 0, 0, 0, 0, 250, 0.0001, 0.0001, 10, 5, sens_c, pot_c};
int vars_control_d[13] = {0, 0, 0, 0, 0, 0, 250, 0.0001, 0.0001, 10, 5, sens_d, pot_d};

// Error
float err_a, err_b, err_c, err_d;
```

Figura 4–5. Creación de variables de control y error.

En la Tabla 2. Vector de control, se encuentra el listado que describe la variable que guarda cada una de las posiciones en este arreglo.

Tabla 2. Vector de control

Posición en el arreglo	***Variable***
0	Valor temporal de posición
1	Posición deseada
2	Velocidad temporal
3	Derivada del error
4	Valor previo del error
5	Integral del error
6	Constante P
7	Constate I
8	Constante D
9	Margen de error máximo
10	Número de muestras a promediar por ciclo
11	Sensor del motor
12	Posición dada por potenciómetro (opcional)

Estas variables se repiten para cada uno de los motores, alguna de ellas cambia dinámicamente por cada ciclo, esto para poder ajustar constantemente la posición de los motores. Además de eso, permite hacer un *tunning* dependiendo del sistema, ya que cada uno puede tener motores y condiciones de uso diferentes.

```
// MOVER DE ACUERDO A UN POTENCIOMETRO
mover_y_controlar_potenciometro_LC(err_a, vars_control_a, ma);
mover_y_controlar_potenciometro_LC(err_b, vars_control_b, mb);
mover_y_controlar_potenciometro_LC(err_c, vars_control_c, mc);
mover_y_controlar_potenciometro_LC(err_d, vars_control_d, md);
```

Figura 4–6. Control de múltiples motores simultáneamente.

```
void mover_y_controlar_posicion_LC(int error, int * vars_control, int * motor) {

  // Extraer variables

  // Verificar limites de posicion
  if (limites(vars_control[1], vars_control[13], vars_control[14]) == true) {
    mover(motor[0], motor[1], motor[2], 0);
    return;
  }

  // Leer y Filtrar señal - Temp pos
  vars_control[0] = filtro_promedios(vars_control[10], vars_control[11]);

  // Calcular el error
  error = vars control[1] - vars control[0];
  // Integral del error
  vars_control[5] = int_error(vars_control[5], error, 50);

  // Derivada del error
  vars_control[3] = error - vars_control[4];

  // Nuevo valor temporal de error
  vars_control[4] = error;

  // Control PID para devolver velocidad del motor
  vars_control[2] = control_PID(error, vars_control[5] , vars_control[3], vars_contro

  // Mover motor
  mover(motor[0], motor[1], motor[2], vars_control[2]);

}
```

Figura 4–7. Función para controlar motor

Como se observa en la Figura 4–7, dentro del mismo código se llaman a diferentes funciones de la misma librería para poder mover el motor. Para poder ver el código completo ver el Anexo B, para poder ver en detalle las funciones mencionadas ver Anexo A. Esta función es de suma importancia para el manejo general de la prótesis, en la Figura 4–8 se resume en bloques su funcionamiento interno.

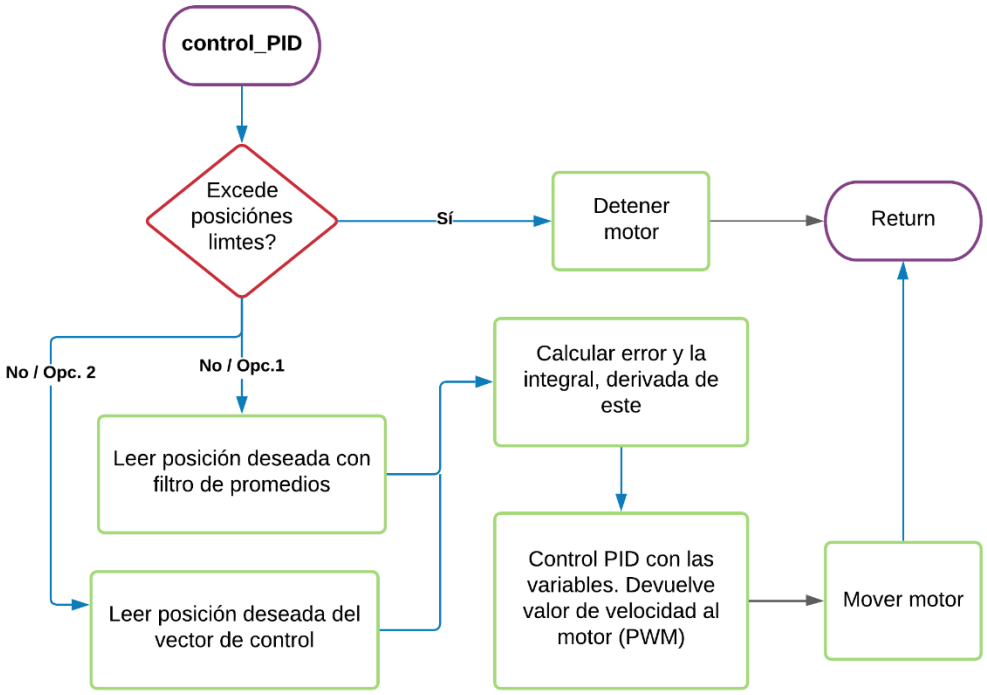


Figura 4–8. Diagrama general del último bloque del diseño de control

4.3. Simulación

Durante el desarrollo del algoritmo se creó una simulación en Proteus usando componentes análogos a los que se tienen usualmente en los montajes de la fundación. Se tuvo que descargar un paquete extra de Arduino.

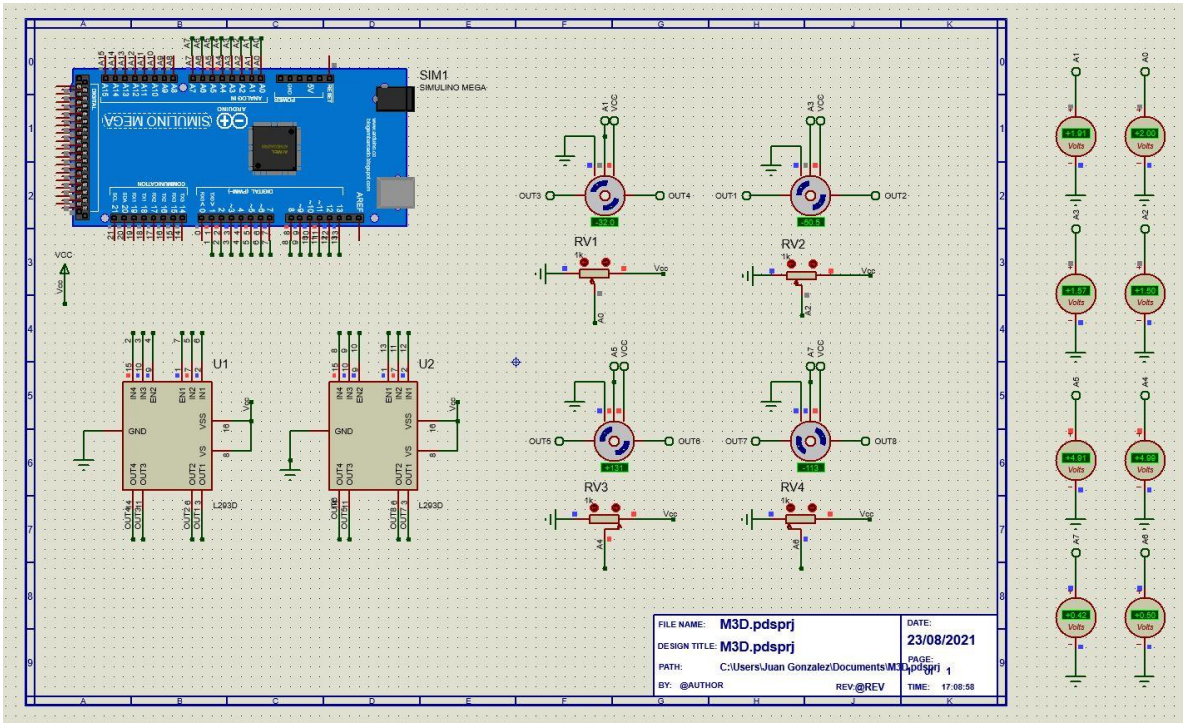


Figura 4–9. Simulación con Implementación de Algoritmo

Para la Figura 4–9, en la parte superior izquierda se observa un Arduino mega, en la esquina inferior izquierda se ven los drivers (cada uno puede manejar dos motores), en la parte central se ven los motores con los potenciómetros que permiten cambiar la posición, y en la parte derecha se observa unos multímetros que comparan el voltaje de referencia con el voltaje obtenido (análogo a la posición).

En la simulación se pueden manejar simultáneamente la posición de varios motores, y teniendo en cuenta visualmente cómo funciona el sistema y los voltímetros, funciona adecuadamente. El error que se observa se debe mayoritariamente a él margen de error que se le da al sistema.

Tabla 3. Desempeño del código en simulación

Valor de referencia (V)	Valor obtenido (V)	Error porcentual (%)
1.91	2.00	4.5
1.57	1.50	4.6
4.91	4.99	1.6
0.42	0.50	16

En la Tabla 3 se puede observar los resultados obtenidos en la simulación usando cuatro motores. Teniendo esto en cuenta, se procede a implementar el código en un montaje real.

4.4. Implementación

La estructura propuesta en la Figura 4–1 para manejar las prótesis, permite una facilidad de poder agregar varios motores, controlar simultáneamente la posición de varios motores, generar posiciones estándar y también posiciones dependientes de la señal del sensor o señal de intención de movimiento (SIMV).

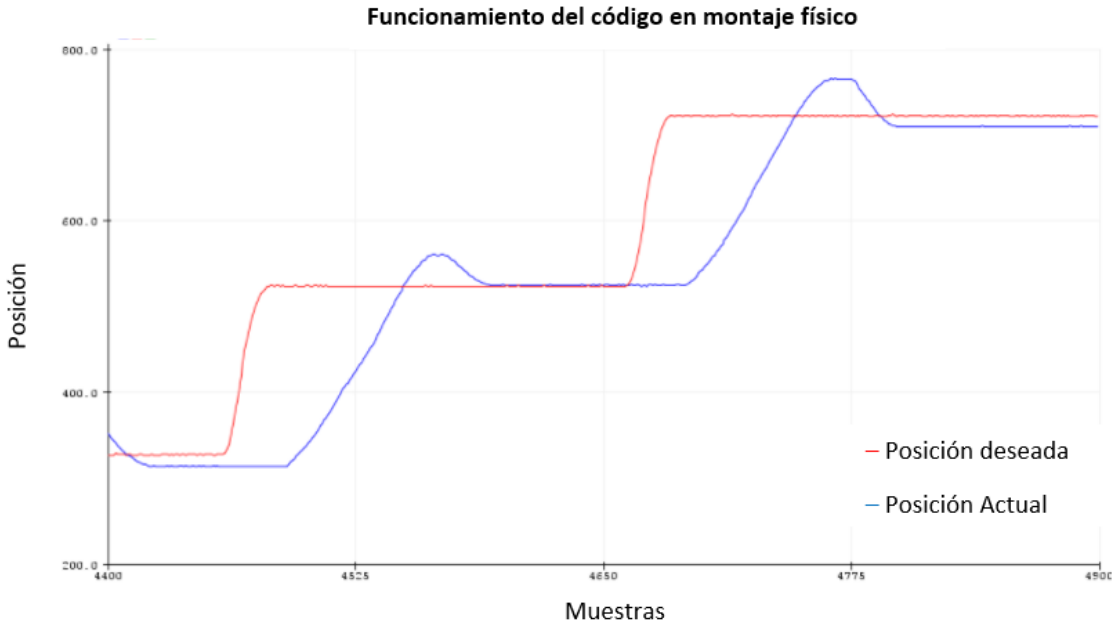
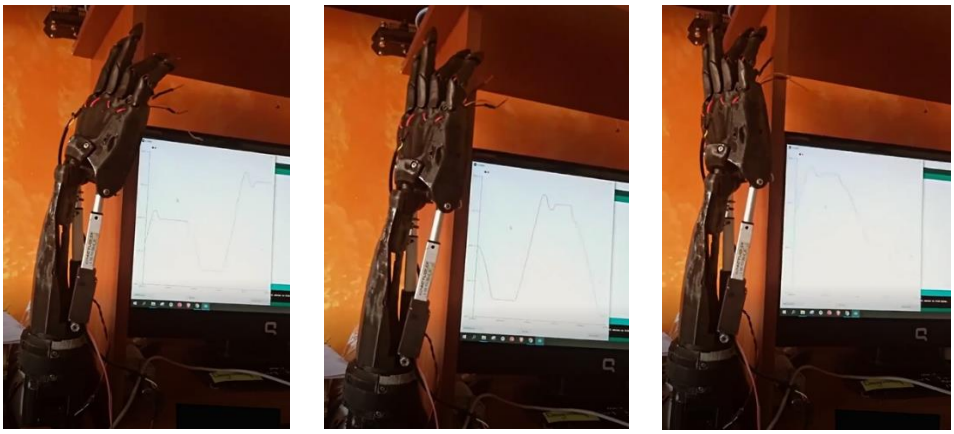


Figura 4–10. Posición deseada vs Actual en sistema real

Al momento de usar el sistema en una prótesis, se empezó usando únicamente un motor, para ver la respuesta de este motor dado el sistema. La respuesta se ve en la Figura 4–10.**Error! Reference source not found.** Este motor hace referencia a uno de los actuadores lineales usados en el antebrazo de la prótesis. Como se observa en la Figura 4–11, la prótesis puede adoptar diferentes posiciones teniendo en cuenta la entrada de un potenciómetro, la cual puede ser reemplazada por un sensor de electromiografía, proximidad, valores constantes en el código u otros. Para ver el funcionamiento en video referir al Anexo 4.



a *b* *c*

Figura 4–11. Prótesis en diferentes posiciones.

5. DISCUSIÓN

Con los resultados obtenidos, se puede observar cómo se generó un sistema capaz de adaptarse a diferentes situaciones teniendo en cuenta las condiciones de la fundación M3D y la naturaleza de su trabajo en el desarrollo de prótesis. Como ventaja, el sistema se puede tanto simular como implementar en un sistema físico, lo cual permite al diseñador de la prótesis experimentar con diferentes parámetros, aunque se pueden mejorar aún más las simulaciones si se tiene en cuenta las condiciones generales.

El modelo planteado del código funcionó satisfactoriamente y es relativamente sencillo de usar, ya que no se requieren muchas líneas de programación para poder manejar los motores y se tiene la documentación completa del código. Aunque se puede modificar para hacer mejoras, el sistema por ahora solo permite usar motores los cuales tengan un sensor de posición, sin embargo, algunas funciones de la librería pueden funcionar sin necesidad de esto (como la función *mover* – 4.1.1).

Una de las ventajas principales del código es permitir el manejo del movimiento de varios motores simultáneamente, lo cual provee más naturalidad en los movimientos del brazo, permite tener niveles continuos de movilidad, mas no rangos absolutos de desplazamiento, como por ejemplo en flexión o extensión. No se puso a prueba el límite en el tiempo de respuesta usando las tarjetas Arduino, pero fluyó con un buen tiempo de respuesta en las pruebas experimentales usando hasta cuatro motores.

El código posee cierta flexibilidad que le ayuda a ajustar el sistema dependiendo de su respuesta. Algunos motores mostraban una respuesta suave y sin sobrepicos en la posición, como motores presentes en los codos, sin embargo, algunos otros tendían a tener un sobre pico significativo e inestabilidad por el mismo sistema, por lo que la implementación de un umbral y las variables de control fue bastante útil, y aunque se pierde

en cierta medida la precisión, no representa un cambio drástico para el alcance de las prótesis en M3D.

En la fundación M3D, muchos de los trabajadores son personas con enfoques de diseño, mas no de electrónica y/o programación, por lo tanto, esto puede ayudar sustancialmente cuando se tengan dichas limitaciones en éste área, más sin embargo se requiere de una pequeña guía o entrenamiento para aprender a usar el sistema.

Otro gran aporte que genera este código es el económico. Por lo general, los servomotores vienen con estos sistemas ya incluidos y no hay necesidad de programar el controlador, pero ellos habían experimentado fallas continuas con estos sistemas, ya que estos se quemaban con facilidad y dejaban de funcionar. Una solución fue crear un código propio código para manejar estos motores con drivers que resistan más potencia y manteniendo el sistema mecánico del motor intacto. Esto reduce sustancialmente el precio total de la prótesis.

Complementando lo anterior, la decisión de hacerlo compatible para Arduino, lo hace muy accesible y fácil de implementar así no se tengan fuertes conocimientos en programación de microcontroladores, lo cual apoya positivamente el programa de la fundación que permite que los beneficiarios también estén trabajando activamente en sus prótesis.

Finalmente, se hicieron pruebas del mismo algoritmo en una tarjeta ESP32, la cual tiene una capacidad mucho mayor a la de los Arduino, ya que posee WIFI, Bluetooth y permitiría en un futuro generar prótesis de una calidad mucho mayor. Durante estas pruebas resultó evidente que necesitaba una función extra no desarrollada dentro de este proyecto (analogRead) pero que con facilidad se podía conseguir de las librerías de Arduino para la ESP32, por lo tanto, no fue un retraso mayor.

Para futuros trabajos sería de gran utilidad ampliar la funcionalidad de algoritmo, incluyendo un sistema extra de seguridad en los movimientos basado en pulsadores o finales de carrera. Consiguientemente, se podría hasta llegar a estandarizar tarjetas para estas prótesis usando este algoritmo.

6. RECOMENDACIONES Y TRABAJOS FUTUROS

Estas librerías pueden ser de gran utilidad asumiendo que se tiene los componentes electrónicos correctamente montados, por lo tanto, sería de gran utilidad tener una PCB diseñada tanto para el Arduino nano como para la tarjeta ESP32, para así poder implementar más rápidamente el código y la electrónica necesaria para las prótesis.

A pesar de que dentro del alcance del código se tiene en cuenta la entrada de los sensores, se puede mejorar el código para poder procesar las señales de estos sensores dependiendo de su naturaleza, ya sea para una señal proveniente de un sensor de EMG o de un sensor infrarrojo. Para esto, se recomienda hacer una breve investigación que tenga en cuenta el componente teórico de cada una de las señales y hacer pruebas reales con los sensores.

Teniendo en cuenta que la idea es reducir al máximo el tiempo de programación del desarrollador de las prótesis, un trabajo futuro de gran utilidad sería hacer una GUI que permita el fácil control y desarrollo del código usando la librería desarrollada, para así poder ampliar el alcance de la prótesis. Para esto se recomienda usar las diferentes herramientas de acceso libre disponibles para aplicaciones de escritorio.

Finalmente, como trabajo futuro, se podría mejorar el funcionamiento del algoritmo de control para abrir el espectro y poder usar diferentes algoritmos, esto con la intención de tener un control mayor sobre cada uno de los motores de la prótesis.

7. CONCLUSIONES

- Se desarrollo una librería en C/C++ para el manejo de las prótesis de la Fundación Materialización 3D. Esta librería es de utilidad siempre y cuando se use tarjetas Arduino, ESP32, o cualquier procesador compatible con el lenguaje de programación.

- Con el uso de la librería se logra reducir considerablemente la complejidad en la programación para manejar los motores teniendo en cuenta el número de líneas de código, además, se incluye la funcionalidad del manejo de múltiples motores simultáneamente si es requerido. No existe límite en el número de motores, depende de la capacidad del hardware.

- Se creó una simulación en Proteus donde se tienen en cuenta tanto los componentes eléctricos como el código desarrollado. El simple hecho de tener una simulación facilita y optimiza el proceso de diseño del algoritmo.

- Se realizaron pruebas físicas del funcionamiento del algoritmo y se demostró un correcto funcionamiento del código sobre la prótesis, sin embargo, es necesario llevar el sistema a la cotidianidad para poder ver la funcionalidad final de este.

- Con el fin de facilitar el acceso a este desarrollo y facilitar su uso, se creó un documente explicando su forma de uso y un repositorio en github para poder acceder a todo el código, las simulaciones, los documentos y las pruebas realizadas.

- Los objetivos se lograron cumplir satisfactoriamente, sin embargo, es necesario que los miembros de la Fundación M3D estén dispuestos a promover y utilizar dicho desarrollo.

REFERENCIAS

[1] L. Resnik *et al.*, “Advanced upper limb prosthetic devices: Implications for upper limb prosthetic rehabilitation,” *Archives of Physical Medicine and Rehabilitation*. 2012, doi: 10.1016/j.apmr.2011.11.010.

[2] D. Van Der Riet, R. Stopforth, G. Bright, and O. Diegel, “An overview and comparison of upper limb prosthetics,” 2013, doi: 10.1109/AFRCON.2013.6757590.

[3] I. Ku, G. K. Lee, C. Y. Park, J. Lee, and E. Jeong, “Clinical outcomes of a low-cost single-channel myoelectric-interface three-dimensional hand prosthesis,” *Arch. Plast. Surg.*, 2019, doi: 10.5999/aps.2018.01375.

[4] J. ten Kate, G. Smit, and P. Breedveld, “3D-printed upper limb prostheses: a review,” *Disability and Rehabilitation: Assistive Technology*. 2017, doi: 10.1080/17483107.2016.1253117.

8. ANEXOS

Anexo 1. Descripción detallada de la librería

Librería de Arduino para Control de Prótesis Fundación M3D

Juan Andrés González Urquijo

I. Introducción

Esta librería tiene como propósito facilitar la programación de los motores en Arduino para manejar una prótesis, con la intención de poder implementar sistemas con varios actuadores. Ya que en la fundación M3D se manejan motores sin drivers, en ocasiones, se dirige parte de la librería a controlar dichos motores usando el controlador L293D.

I) *Contenido*

Para crear una librería útil en arduino, es necesario tener tres (3) archivos diferentes.

- Main.ino : Código donde se hace llamado de las funciones y utilidades de la librería
- lib_motor.cpp: Definir las funciones y el código de las mismas
- lib_motor.h : Definir nuevamente las funciones, pero sin el código.

Funciones de la librería. Recordar que la palabra antes del nombre de la función (void,int,double...) indica que tipo de variable devuelve la función. Si es void, no devuelve nada. Los archivos se pueden encontrar en: *https://github.com/JuanGonzalezU/Code-Arm-prosthesis*

i) void mover

a. Descripción: Mueve 1 motor asumiendo que está conectado a un controlador L293D. Esta función mantiene el movimiento hasta que se llame nuevamente la misma función o se detenga de otra manera.
b. Argumentos (entradas):
    i. int m1 : Salida digital del Arduino y entrada a el pin 2 ó 10 del L293D
    ii. int m2 : Salida digital del Arduino y entrada a el pin 7 ó 15 del L293D
    iii. int v : Valor de velocidad del motor. De –255 a –1 es en una dirección, de 0 a 255 es en dirección opuesta. La velocidad esta definida por el valor absoluto del valor de v, y la dirección por el signo.
c. Ejemplo de uso: *mover(2,3, 4,-255)*;

ii) *int leer_analogo*

a. Descripción: Lee el puerto analogo e imprime los valores en el Serial plotter
b. Argumentos (entradas):
    i. int puerto : Puerto analogo que se desea leer (A0,A1,A2...)
    ii. Bool imprimir : ‘true’ si se desea imprimir y ‘false’ si no.
c. Ejemplo de uso: *leer_analogo(A0, false)*

iii) int filtro_promedios

a. Descripción: Adquire datos de un puerto análogo y hace un filtro de promedios.
b. Argumentos (entradas):
    i. int muestas : Numero de muestras que se usan para hacer el filtro (promediar)
    ii. int puerto : Puerto del cual se toman los datos
c. Ejemplo de uso: *tmp_pos = filtro_promedios(10, A0);*

iv) int sgn

a. Descripción: Devuelve el signo del número de entada
b. Argumentos (entradas):
    i. int x : Numero entero
c. Ejemplo de uso: sgn(-5)

v) Void inicializar_motor_pot

a. Descripción: Defines los pines de entrada, salida del que van al driver del motor y del análogo del potenciómetro, que marcaría la posición deseada del motor.
b. Argumentos:
    i. Int * motor:
        1. int m1 : Salida digital del Arduino y entrada a el pin 2 ó 10 del L293D
        2. int m2 : Salida digital del Arduino y entrada a el pin 7 ó 15 del L293D
        3. int v : Valor de velocidad del motor. De –255 a –1 es en una dirección, de 0 a 255 es en dirección opuesta. La velocidad esta definida por el valor absoluto del valor de v, y la dirección por el signo.
    ii. Int sensor : Entrada análoga del sensor del motor que define la posición REAL.
    iii. Int pot : Entrada análoga del potenciometro del motor que define la posición DESEADA.

vi) Void inicializar_motor

a. Descripción: Defines los pines de entrada, salida del que van al driver del motor.
b. Argumentos:
    i. Int * motor:
        1. int m1 : Salida digital del Arduino y entrada a el pin 2 ó 10 del L293D
        2. int m2 : Salida digital del Arduino y entrada a el pin 7 ó 15 del L293D
        3. int v : Valor de velocidad del motor. De –255 a –1 es en una dirección, de 0 a 255 es en dirección opuesta. La velocidad

esta definida por el valor absoluto del valor de v, y la dirección por el signo.

ii. Int sensor : Entrada análoga del sensor del motor que define la posición REAL.

vii) Int control_p

a. Descripción: Implementa un control tipo p y permite un margen de error. Esta función devuelve la velocidad (que se puede usar como argumento de la función *mover*)
b. Argumentos (entradas):
    i. int error : Error entre la posición deseada y la posición actual
    ii. int margen : Margen de error aceptado, donde el controlador tipo P dejará de funcionar.
    iii. int P : Constante Kp

viii) Int int_error

a. Descripción: Calcula la integral del error y devuelve el valor. Si el error es bajo, se reinicia el valor de la integral a 0.
b. Argumentos (entradas):
    i. Int int_error: Integral del error
    ii. Int new_err: Nuevo valor del error
    iii. Margen: error al cual se reinicia el valor de la integral

ix) Int control_PI

a. Descripción: Hace un contrl tipo PI. Devuelve la velocidad del motor.
b. Argumentos:
    i. Int error : Error entre la posición deseada y la posición actual
    ii. Int int_error : Integral del error
    iii. Int I : Constante Ki
    iv. Int margen: Margen de error

x) Int control_PID

a. Descripción: Hace un control tipo PID. Devuelve la velocidad del motor. Esta función solo multiplica el signo del error por P, ya que dado el montaje esto se tuvo que cambiar. Sin embargo, se puede modificar manualmente, cambiando en la función
*" return sgn(error) * P + int_err * I + der_err * D "* por *" return error * P + int_err * I + der_err * D "*
b. Argumentos:
    i. Int error : Error entre la posición deseada y la posición actual
    ii. Int int_error : Integral del error
    iii. Int der_error : Derivada del error
    iv. Int P : Constante Kp

v. Int I : Constante Ki
vi. Int D : Consante Kd.
vii. Int margen : Margen del error aceptado

xi) void mover_y_controlar_posicion_LC

a. Descripción: Mueve el motor y controla la posición en lazo cerrado usando un controlador tipo PID. Es necesario enviar en la variable de control la posición (posición 2)
b. Argumentos :
    i. Int error : Error entre la posición deseada y la posición actual
    ii. Int * vars_control : Arreglo de variables de control (mirar ejemplo 1)
        1. Valor temporal de posición
        2. Posición deseada
        3. Velocidad temporal
        4. Derivada del error
        5. Valor anterior de error
        6. Integral del error
        7. constante P
        8. constante I
        9. constante D
        10. margen de error máximo (0 1023)
        11. Numero de muestras a promediar por ciclo
        12. Sensor del motor
        13. Posición deseada dada por un potenciómetro
    iii. Int * motor : Arreglo de variables referentes al motor. Mismos argumentos que los de la función *mover*.

xii) mover_y_controlar_potenciometro_LC:

a. Descripción: Descripción: Mueve el motor y controla la posición en lazo cerrado usando un controlador tipo PID. El motor sigue la curva del potenciómetro.
b. Argumentos : Int error : Error entre la posición deseada y la posición actual
    i. Int * vars_control : Arreglo de variables de control (mirar ejemplo 1)
        1. Valor temporal de posición
        2. Posición deseada
        3. Velocidad temporal
        4. Derivada del error
        5. Valor anterior de error
        6. Integral del error
        7. constante P
        8. constante I
        9. constante D
        10. margen de error máximo (0 1023)
        11. Numero de muestras a promediar por ciclo

12. Sensor del motor
13. Posición deseada dada por un potenciómetro

ii. Int * motor : Arreglo de variables referentes al motor. Mismos argumentos que los de la función *mover*

xiii) Bool limites:

a. Descripción: Revisa los límites de posición dados por el diseñador con la posición del motor y devuelve una respuesta booleana para continuar o parar el movimiento. Para eso hace el chequeo con la posición deseada. Si no cumple con los limites superiores e inferiores devuelve un TRUE, si no devuelve FALSE.
b. Argumentos:
    i. Int des_pos: posición deseada
    ii. Int max_pos: Máxima posición de movilidad
    iii. Int min_pos: Mínima posición de movilidad

Anexo 2. Link al código, versiones y adaptaciones del código

El código completo, las versiones, y la adaptación para la ESP32, se pueden encontrar en el siguiente link: *https://github.com/JuanGonzalezU/Code-Arm-prosthesis*

Anexo 3. Codigo completo multiples motores

Multi_motor.ino

```
1  //Codigo para controlar motores teniendo en cuenta la libreria creada
2
3  #include "lib_motor.h"
4
5  // Definir variables ---------------------------------------------
6
7  //Salida al motor
8  const int ma[] = {2, 3, 4};
9  const int mb[] = {5, 6, 7};
10 const int mc[] = {8, 9, 10};
11 const int md[] = {11, 12, 13};
12
13 // Entradas
14 const int  sens_a = A1, pot_a = A0;
15 const int  sens_b = A3, pot_b = A2;
16 const int  sens_c = A5, pot_c = A4;
17 const int  sens_d = A7, pot_d = A6;
18
19 // Inicalización -------------------------------------------------
20
21 void setup() {
22
23   // Open serial
24   Serial.begin(9600);
25
26   // Inicializar pines del motor
27   inicializar_motor(ma, sens_a, pot_a);
28   inicializar_motor(mb, sens_b, pot_b);
29   inicializar_motor(mc, sens_c, pot_c);
30   inicializar_motor(md, sens_d, pot_d);
31 }
```

```
32
33  // Variables de control  ----------------------------------------
34
35  int vars_control_a[13] = {0, 0, 0, 0, 0, 0, 200, 0.0001, 0.0001, 10, 5, sens_a, pot_a};
36  int vars_control_b[13] = {0, 0, 0, 0, 0, 0, 250, 0.0001, 0.0001, 10, 5, sens_b, pot_b};
37  int vars_control_c[13] = {0, 0, 0, 0, 0, 0, 250, 0.0001, 0.0001, 10, 5, sens_c, pot_c};
38  int vars_control_d[13] = {0, 0, 0, 0, 0, 0, 250, 0.0001, 0.0001, 10, 5, sens_d, pot_d};
39
40  // Error
41  float err_a, err_b, err_c, err_d;
42
43  // Ciclo principal -------------------------------------------------
44
45  void loop() {
46
47    // MOVER DE ACUERDO A UN POTENCIOMETRO
48    mover_y_controlar_potenciometro_LC(err_a, vars_control_a, ma);
49    mover_y_controlar_potenciometro_LC(err_b, vars_control_b, mb);
50    mover_y_controlar_potenciometro_LC(err_c, vars_control_c, mc);
51    mover_y_controlar_potenciometro_LC(err_d, vars_control_d, md);
52
53
54    /*
55    vars_control_a[1] = 750;
56    vars_control_b[1] = 750;
57    vars_control_c[1] = 750;
58    vars_control_d[1] = 750;
59
60    mover_y_controlar_posicion_LC(err_a, vars_control_a, ma);
61    mover_y_controlar_posicion_LC(err_b, vars_control_b, mb);
62    mover_y_controlar_posicion_LC(err_c, vars_control_c, mc);
63    mover_y_controlar_posicion_LC(err_d, vars_control_d, md);
64    */
```

Anexo 4. Funcionamiento del código en montaje real

https://youtu.be/kxT71qBOaB8