UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE CUENCA

CARRERA DE ELECTRÓNICA Y AUTOMATIZACIÓN

DESARROLLO DE UNA PROPUESTA PARA LA EMULACIÓN DE LOS MOVIMIENTOS DE MUÑECA DE UN BRAZO PROTÉSICO BASADO EN UN MECANISMO CON TRES GRADOS DE LIBERTAD.

Trabajo de titulación previo a la obtención del título de Ingeniero en Electrónica

AUTOR: MARTÍN SANTIAGO BRAVO JIMÉNEZ

TUTOR: ING. LUIS JAVIER SERPA ANDRADE, Mgtr.

Cuenca – Ecuador

2024

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Yo, Martín Santiago Bravo Jiménez con documento de identificación N°0106047327; manifiesto que:

Soy autor y responsable del presente trabajo; y, autorizo a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Cuenca, 2 de agosto de 2024

Atentamente,

Martín Santiago Bravo Jiménez
0106047327

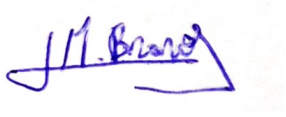
CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Yo, Martín Santiago Bravo Jiménez con documento de identificación N° 0106047327, expreso mi voluntad y por medio del presente documento cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy autor del Proyecto Técnico: "Desarrollo de una propuesta para la emulación de los movimientos de muñeca de un brazo protésico basado en un mecanismo con tres grados de libertad.", el cual ha sido desarrollado para optar por el título de: Ingeniero en Electrónica, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribimos este documento en el momento que hacemos la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Cuenca, 2 de agosto de 2024

Atentamente,



Martín Santiago Bravo Jiménez
0106047327

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Luis Javier Serpa Andrade con documento de identificación N° 0103779096, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: DESARROLLO DE UNA PROPUESTA PARA LA EMULACIÓN DE LOS MOVIMIENTOS DE MUÑECA DE UN BRAZO PROTÉSICO BASADO EN UN MECANISMO CON TRES GRADOS DE LIBERTAD., realizado por Martín Santiago Bravo Jiménez con documento de identificación N° 0106047327, obteniendo como resultado final el trabajo de titulación bajo la opción Proyecto Técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Cuenca, 2 de agosto de 2024

Atentamente,

Ing, Luis Javier Serpa Andrade, Mgtr.
0103779096

AGRADECIMIENTOS

Quisiera expresar mi agradecimiento a mis padres por financiar mis estudios y por su apoyo. También deseo agradecer a Henry Cumbe por su valiosa ayuda con la parte mecánica del sistema.

Martín Santiago Bravo Jiménez

DEDICATORIAS

Martín Santiago Bravo Jiménez

Este trabajo se lo dedico a mis padres, que ha pesar no ser perfectos, me apoyaron durante mi carrera universitaria.

Resumen

En este trabajo de titulación se ha desarrollado una propuesta para la emulación de los movimientos de muñeca de un brazo protésico, basado en un mecanismo con tres grados de libertad. Este proyecto se enmarca en la necesidad de mejorar la funcionalidad y precisión de las prótesis de muñeca, proporcionando una mayor movilidad y naturalidad en los movimientos para los usuarios. Se investigaron diversos tipos de actuadores y mecanismos utilizados en prótesis de muñeca, incluyendo mecanismos seriales y paralelos. Se determinó que los mecanismos paralelos con tres grados de libertad ofrecen una mayor precisión y rigidez, además de una mejor capacidad de carga en comparación con los mecanismos seriales. En este proyecto se trabajó con un mecanismo paralelo de rótula activa basado en engranajes esféricos, conocido como ABENICS (Active Ball Joint Mechanism With Three-DoF Based on Spherical Gear Meshing). El desarrollo del proyecto incluyó el diseño e implementación de una placa de circuito impreso (PCB) y la programación de un ESP32 para controlar los movimientos del mecanismo. También se desarrolló una página web para la gestión y control del efector final, permitiendo así la interacción en tiempo real y la configuración de los parámetros necesarios para el funcionamiento del sistema. Finalmente, cabe decir que se tuvieron problemas con la parte mecánica lo cual limito en ciertos aspectos el desarrollo de este trabajo.

Palabras clave: ABENICS; Prótesis; DOF; Muñeca; Actuadores; PCB

Abstract

In this degree work, a proposal has been developed for the emulation of wrist movements for a prosthetic arm, based on a mechanism with three degrees of freedom. This project is framed in the need to improve the functionality and precision of wrist prostheses, providing greater mobility and natural movements for users. Several types of actuators and mechanisms used in wrist prostheses were investigated, including serial and parallel mechanisms. It was determined that parallel mechanisms with three degrees of freedom offer greater accuracy and stiffness, as well as better load capacity compared to serial mechanisms. In this project, we worked with an Active Ball Joint Mechanism With Three-DoF Based on Spherical Gear Meshing, known as ABENICS . The project's development included the design and implementation of a printed circuit board (PCB) and the programming of an ESP32 to control the movements of the mechanism. A web page was also developed for the management and control of the end effector, allowing real-time interaction and configuration of the necessary parameters for the system's operation.Finally, it is worth noting that there were issues with the mechanical parts, which limited the development of certain aspects of this project.

Keywords: ABENICS; Prosthetics; DOF; Wrist; Actuators; PCB

Antecedentes

En el ámbito de la investigación sobre prótesis y dispositivos biomédicos, se han llevado a cabo numerosos estudios que exploran diversas tecnologías aplicadas al movimiento de muñeca. Los actuadores para prótesis de muñeca se pueden clasificar según su tipo de mecanismo, método de actuación, y sus grados de libertad (DOF, del inglés *Degrees Of Freedom*). Los actuadores más comunes pueden poseer un mecanismo serial o paralelo. Las prótesis con actuadores seriales pasivos y con un grado de libertad han sido las que han dominado en los últimos 80 años debido a su compactes, simplicidad mecánica, y bajo peso. No obstante, el problema que poseen los actuadores pasivos, es que requieren de fuerzas externas para ajustar el efector final [1]. Por otro lado, las prótesis con actuadores activos basan su movimiento mediante fuerzas ejercidas por motores o el mismo cuerpo. De la misma manera, estas prótesis permiten automatizar y realizar movimientos coordinados en comparación con las prótesis pasivas. Por esta razón, este proyecto se enfocará en actuadores activos.

La muñeca de un ser humano es capaz de realizar tres movimientos: flexión/extensión, desviación radial/ulnar y pronación/supinación. En términos de actuadores para muñeca con un grado de libertad, el movimiento rotatorio es el que en general se emplea. Un ejemplo de este mecanismo es el usado en [2]. En cuanto a actuadores paralelos con un grado de libertad, estos no son usados frecuentemente debido a la simplicidad mecánica del serial. Por otra parte, con respecto a mecanismos con dos DOF, el expuesto en [3] es el que resalta, puesto que es capaz de realizar los tres movimientos de muñeca. Ahora, si se habla de actuadores con 3 DOF se puede mencionar el presentado en [4] el cual ubica tres rotadores en serie (RRR) con ejes en diferentes orientaciones. Y por último, el actuador paralelo con tres DOF más conocido es el Agile Eye [5]. Este, es un mecanismo 3RRR simétrico, con juntas R de

base accionadas con motores rotativos y al igual que el anterior, es capaz de realizar los tres movimientos [1].

Si analizamos estos actuadores, el problema con los mecanismos que no poseen tres grados de libertad es que no son capaces de realizar los tres movimientos posibles de la muñeca libremente. El actuador usado en [3], posee dos grados de libertad y es capaz de realizar los tres movimientos. No obstante, la desviación ulnar/radial esta limitado a la posición de la muñeca en términos de flexión/extensión. Si se pasa a los mecanismos seriales, estos poseen una mayor área de trabajo, son más robustos y se pueden diseñar con menos componentes en comparación con los paralelos. El problema con los mecanismos seriales es que son vulnerables a sufrir deformaciones y errores acumulativos lo cual produce una menor precisión que los mecanismos paralelos. Por otro lado, las ventajas de los mecanismos paralelos son que poseen mayor precisión al evadir errores acumulativos y disponen de una estructura más estable lo cual permite mayor rigidez y capacidad de carga [6].

Justificación

Según [7], hasta el 2019, aproximadamente 19 millones de personas han sufrido de algún tipo de amputación en las extremidades superiores. Estos problemas conllevan importantes discapacidades para quienes los padecen, lo cual puede tener un impacto psicológico considerable. Por consiguiente, las prótesis de extremidades superiores desempeñan un papel crucial al facilitar la reintegración de estas personas en la sociedad y acercarlas a una vida lo más normal posible. Una parte esencial para el diseño de una prótesis de muñeca es el actuador, dado que proporciona los movimientos necesarios para replicar las funciones naturales de la articulación. Desarrollar actuadores más avanzados e innovadores permitirá mejorar la funcionalidad y la movilidad de las prótesis, lo que beneficiará directamente a los usuarios al permitirles realizar una gama más amplia de actividades con mayor precisión y naturalidad. De la misma manera, el desarrollo de actuadores innovadores para prótesis de muñeca no solo tiene beneficios médicos y funcionales, sino que también puede tener un impacto significativo en términos económicos y sociales. Para finalizar, se debe tener en claro que una mejora en la funcionalidad y la adaptabilidad de las prótesis puede reducir los costos asociados con la atención médica a largo plazo y mejorar la participación laboral y la calidad de vida de las personas.

Como se desea realizar los tres movimientos posibles de muñeca sin ninguna restricción, el actuador a trabajar debe poseer tres grados de libertad. Con respecto a que mecanismo usar, se concluye que un mecanismo paralelo será la mejor opción. Las razones que llevaron a esta conclusión se debe a que este mecanismo posee una mejor precisión y no requiere de mucho espacio longitudinal en comparación con mecanismos seriales que no usan acoplamientos diferenciales. De la misma manera, los mecanismos paralelos poseen mayor rigidez y capacidad de carga. Por último, una

de las desventajas de los actuadores paralelos es el área de trabajo, no obstante, como se está hablando de movimientos de muñeca para una persona, el área de trabajo no debe ser amplio [1], [6].

El presente trabajo, parte de un mecanismo paralelo de rótula activa con tres grados de libertad basado en engranajes esféricos (ABENICS, del ingles *Active Ball Joint Mechanism With Three-DoF Based on Spherical Gear Meshings*), desarrollado por un grupo de la materia de Teoría de Control. El proyecto en cuestión consiste en adaptar y controlar este sistema utilizando un microcontrolador, con el fin de dotar al mecanismo de la capacidad de realizar los tres movimientos posibles de muñeca, en el espacio de trabajo designado. La razón de esta implementación se debe a que si se quiere adaptar este actuador a una prótesis de muñeca, entonces este debe ser capaz de ser controlado mediante un circuito pequeño y liviano. Este no puede ser controlado mediante un PLC por ejemplo, por ende la importancia de este trabajo.

Objetivos

Objetivo General

- Desarrollar una propuesta para la emulación de los movimientos de muñeca de un brazo protésico basado en un mecanismo con tres grados de libertad.

Objetivos específicos:

- Describir las características y tecnologías propuestas y utilizadas para prótesis de muñeca.
- Desarrollar una interfaz para el control de los movimientos del mecanismo.
- Realizar pruebas de comunicación entre la pagina web y el Esp32, así como la implementación de los movimientos del mecanismo.

Introducción

Según [7], hasta el 2019, aproximadamente 19 millones de personas han sufrido de algún tipo de amputación en las extremidades superiores. Estos problemas conllevan importantes discapacidades para quienes los padecen, lo cual puede tener un impacto psicológico considerable. Por consiguiente, las prótesis de extremidades superiores desempeñan un papel crucial al facilitar la reintegración de estas personas en la sociedad y acercarlas a una vida lo más normal posible.

Para diseñar una prótesis efectiva, es fundamental definir claramente su mecanismo, es decir, cómo se van a ejecutar los movimientos. Este proyecto de titulación se centra en el control de un mecanismo de rótula activa con tres grados de libertad, basado en engranajes esféricos. Este mecanismo usa motores a pasos para controlar el efector final y esta basado en el mecanismo ABENICS. Este mecanismo cuenta con un engranaje esférico cruzado y dos engranajes monopolos, el cual el engranaje esférico cruzado es el que posee el efector final. El objetivo principal de este trabajo radica en demostrar que este mecanismo es capaz de reproducir los tres movimientos posibles de la muñeca.

Capítulo 1

Fundamentación Teórica

Este capítulo tiene como objetivo exponer toda la información relevante sobre el proyecto a realizarse. Se dará a conocer la teoría y nomenclatura de los engranajes rectos, esenciales en la transmisión de potencia mecánica. Además, se analizará el innovador mecanismo de rótula activa con tres grados de libertad, basado en engranajes esféricos, y su funcionamiento. Asimismo, se proporcionará información sobre la fuente de alimentación del sistema y una visión detallada del ESP32 Heltec WiFi Kit 32 v2, destacando sus características y funcionalidades. También se abordará el análisis de los motores a paso Nema 23 y sus controladores TB6600, esenciales para el control preciso de movimiento en sistemas automatizados. De la misma manera, se expondrá las librerías que se usaron para la programación. Este capítulo tiene como finalidad proporcionar una comprensión profunda de los componentes y principios teóricos involucrados, asegurando una base sólida para la implementación y desarrollo del proyecto.

Muñeca y sus movimientos

La muñeca humana es un mecanismo el cual permite que la mano sea colocada en el espacio [8]. La muñeca posee dos grados de libertad: flexión/extensión y desviación radial/ulnar. No obstante, cuando un persona pierde la muñeca, entra en juego otro movimiento: pronación/supinación. El responsable de este movimiento es el antebrazo, pero al perder la muñeca, también se pierde esta acción [2]. Por

ende, se concluye que una prótesis de muñeca posee tres grados de libertad: flexión/extensión, desviación radial/ulnar, y pronación/supinación. La figura 1.1 expone estos movimientos.

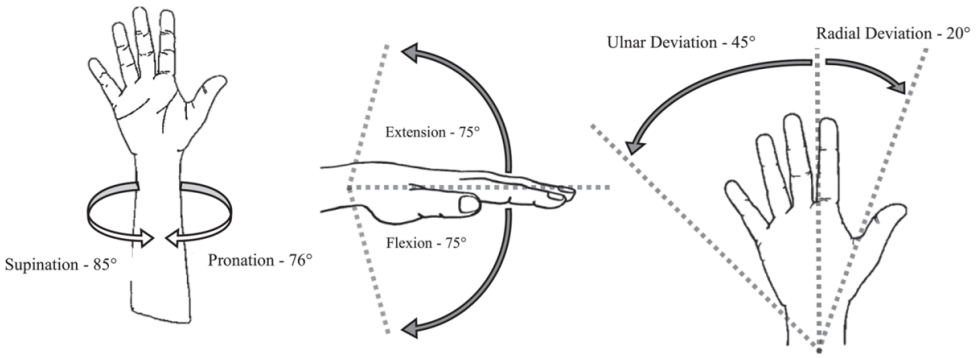


Figura 1.1: Movimientos de la muñeca humana y sus rangos [9].

Mecanismo de rótula activa con tres grados de libertad basados en engranajes esféricos

Un mecanismo de rótula activa con tres grados de libertad basados en engranajes esféricos o ABENICS, es un actuador el cual está compuesto por una engranaje esférico transverzal (CS-gear, del ingles *Cross Spherical Gear*) y dos engranajes monopolos (MP-gear, del ingles *Monopole Gear*). Los MP-gears son los encargados de contralar el CS-Gear.

CS-gear

El engranaje esférico transversal se fabrica cortando dos patrones de dientes simétricos en el eje de una esfera [10]. Primero se graba la estructura del diente en la superficie de la esfera alrededor del eje x como un torno de procesamiento. Esta estructura se basa en el perfil de un engranaje con involuta proyectado en el plano x-y formado por el eje x y el y. Luego, grabando alrededor del eje y, se forma una estructura dental esférica en cuadratura alrededor de ambos ejes en la superficie esférica. En la figura 1.2, la parte (a) muestra el material de la esfera antes de grabar los dientes. En la parte (b), se muestra la estructura dental formada alrededor del eje x. Por ultimo, en

la parte (c) se puede observar el engranaje esférico transversal, donde se han grabado los dientes alrededor del eje x (estructura axial A) y el eje y (estructura axial B) [11]. Cabe decir que el engranaje esférico de la figura **??**(b), posee dos polos mientras que el (c) posee cuatro.

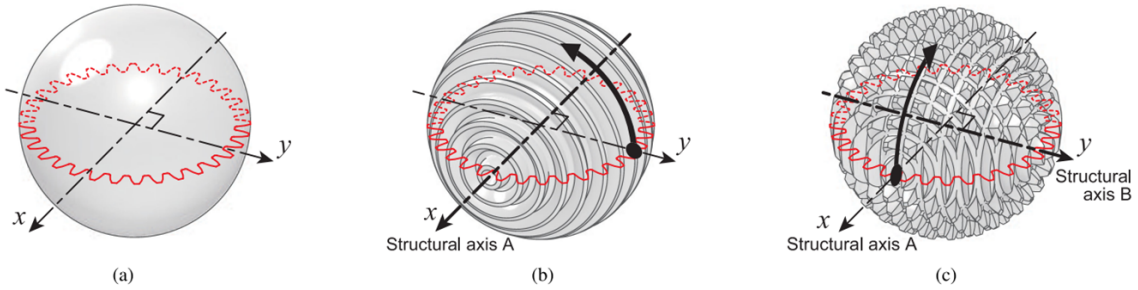


Figura 1.2: Proceso de formación de un engranaje Esférico [11].

El módulo m_{sph} del perfil no posee restricciones. No obstante, el número de dientes z_{sph} está sujeto a dos restricciones: El numero de dientes debe ser par y los ejes x e y están ubicados en el centro del valle o el pico del diente [11].

MP-gear

El proceso de creación de engranajes monopolos implica cortar un patrón de dientes en una superficie cilíndrica que sea capaz de engranar el Cs-gear [10]. El modulo m_{mpl} es igual que el anterior. No obstante, el numero de dientes Z_{mpl} es la mitad que el del Cs-gear.

Como los dos polos del engranaje esférico de la figura 1.2(b) son simétricos, un MP-gear con la mitad de dientes y un solo polo puede engranar ilimitadamente con este engranaje. Asimismo, un polo de este engranaje puede corresponder a dos polos del Cs-gear. Por ultimo, el gran ancho de dientes del Mp-gear limita el Cs-gear, estabilizando así el movimiento de acoplamiento. [11].

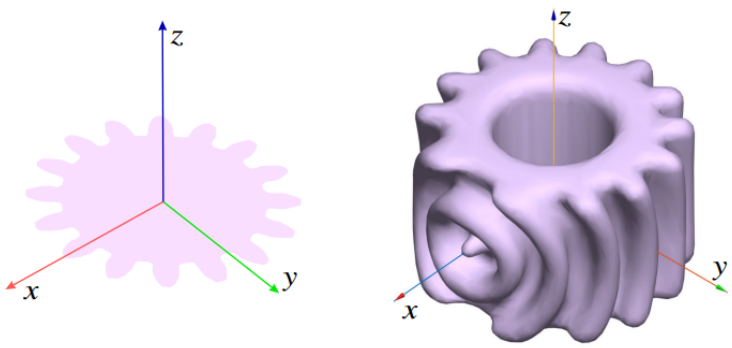


Figura 1.3: Engranahe Monopolo [10].

En la Figura 1.3 se presenta el MP-gear en una representación bidimensional para proporcionar una idea clara de su estructura. Además, se muestra el MP-gear en tres dimensiones para ofrecer una perspectiva completa de su diseño.

Posibles movimientos del mecanismo

El mecanismo es capaz de realizar tres movimientos: Roll, pitch, y yaw. En la figura 1.4 se muestra la configuración del mecanismo y los emparejamientos de los MP-gears con los polos del Cs-gear. De la misma manera, se muestra los movimientos correspondientes de los MP-gears para realizar el pitch y el roll.

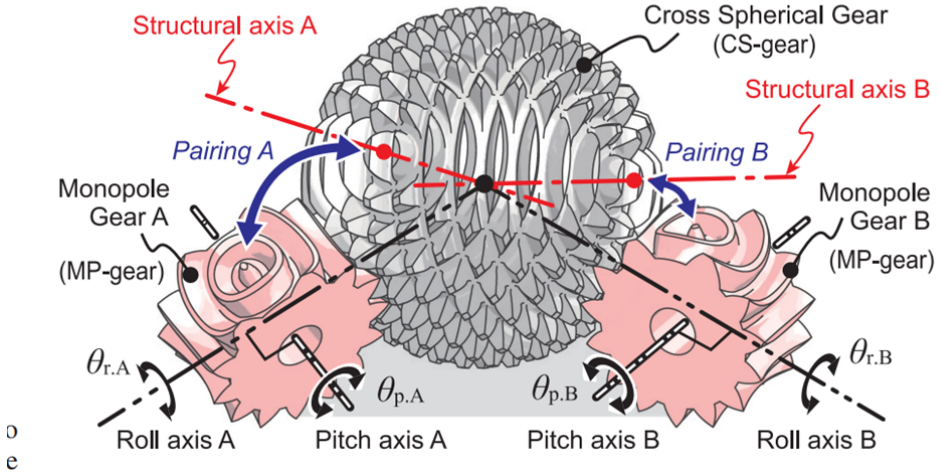


Figura 1.4: Configuración y movimientos del mecanismo [11].

Para realizar el roll, un MP-gear debe estar ubicado en la trayectoria de los

4 polos del Cs-gear mientras que el otro MP-gear estará a 90 grados del primero. Este movimiento puede ser realizado por cualquiera de los MP-gear, con la condición que uno quede inactivo. Por otro lado, para realizar el pitch, un Mp-gear debe ser estacionario y alineado a uno de sus dos polos correspondientes mientras que el otro engranaje sera el que se mueva y deberá pasar por sus dos polos. Para realizar este movimiento, solo se tendrá que accionar el segundo MP-gear. Por último, para realizar el yaw, un Mp-gear no se tiene que mover y puede comenzar en la misma posición que el MP-gear estacionario del pitch. Con respecto al otro MP-gear, este deberá estar inclinado y se moverá sobre la trayectoria de sus polos.

Fuente de Alimentación

Para este proyecto, se usará una fuente de alimentación de 24V. Esta fuente tiene una potencia nominal de 350w y de 14.5A para una salida de 24V. Esta fuente estará directamente conectada a un toma corriente. La salida de 24V se conectara a un regulador de voltaje variable y se ajustará su salida a 5V. Una vez hecho esto, se usara ese voltaje para alimentar el ESP32.

ESP32 (Heltec WiFi Kit 32 v2)

WiFi Kit 32 es una placa de desarrollo IoT clásica diseñada y producida por Heltec Automation(TM), Es un producto altamente integrado basado en ESP32 (incluye Wi-Fi y BLE), batería Li-Po sistema de gestión. También se incluye OLED de 0,96 pulgadas. Es la mejor opción para ciudades inteligentes, granjas inteligentes, hogares inteligentes y fabricantes de IoT [12].

Características del Producto

El WiFi Kit 32 cuenta con un chip maestro ESP32 con un procesador dual-core a 240MHz y 520KB de SRAM interno, acompañado de un chip USB a serie CP2102. Ofrece conectividad Wi-Fi 802.11 b/g/n y Bluetooth V4.2 BR/EDR y LE. La pantalla OLED de 0.96 pulgadas permite visualizar información fácilmente. En términos de

hardware, dispone de 3 UART, 2 SPI, 2 I2C, 1 I2S, 18 entradas ADC de 12 bits, 2 salidas DAC de 8 bits, 22 GPIO y 6 GPI. La memoria flash SPI es de 4MB. Las interfaces incluyen un puerto Micro USB y 18 x 2.54 pin x 2. Su modo de alimentación puede ser mediante USB, batería de litio de 3.7V, o por los pines de 5V y 3.3V. Las dimensiones del dispositivo son 51 x 25.5 x 10.6 mm y opera en un rango de temperatura de -20 a 70 °C [12].

Motores a paso y sus respectivos drivers

Motores a paso Nema 23

Los motores a paso son de gran utilidad debido a que se puede tener una mejor precisión en comparación con otros motores. Las dimensiones de los motores que se usaran son de 57*76mm. Estos son capaces de llegar hasta 1.8Nm de torque y su corriente nominal es de 3A.

Drivers TB6600

El TB6600 es un driver diseñado para controlar motores a paso. Este driver posee tres parámetros de control: PULL, DIR, ENA los cuales pueden ser positivos o negativos. El parámetro DIR es el que especifica la dirección, mientras que el Pull es el que da los pulsos al motor. El ENA se puede dejar desconectado. Por otro lado, los parámetros A y B son los que se conectaran al motor de pasos [13].

Existen dos tipos de conexiones de los parámetros de control, ánodo común y cátodo común. Si se realizar una conexión ánodo común, los pines positivos se conectaran directamente a 5V (Vcc del controlador), mientras que los negativos se conectaran a pines del controlador. Si la alimentación es mayor a 5V, se tendrá que añadir una resistencia R a la salida de cada pin. Para realizar una conexión cátodo común, los positivos se conectaran pines del controlador mientras que los negativos a tierra (GND del controlador) [13].

El dip switch presente en el driver tiene como función modificar los micro-pasos necesarios para realizar un giro completo del eje y la corriente que se

desea aplicar. Para establecer los pasos se tendrá que modificar los 3 primeros switchs, mientras que para controlar la corriente se tendrá que manipular los siguientes tres [13].

Librerías Usadas

- AccelStepper: Es una biblioteca diseñada para controlar motores paso a paso. Proporciona una interfaz avanzada para manejar motores paso a paso de una manera más eficiente y flexible que otras bibliotecas estándar.
- WIFI: Esta librería es utilizada para conectar un dispositivo compatible con WiFi (como el ESP32 o el ESP8266) a una red WiFi. Proporciona funciones para configurar el modo WiFi (cliente o punto de acceso), conectarse a redes WiFi, y manejar eventos de conexión/desconexión.
- ESPAsyncWebServer: Esta es una librería para crear un servidor web asincrónico en dispositivos ESP8266 y ESP32. A diferencia de los servidores web sincrónicos, permite manejar múltiples conexiones al mismo tiempo sin bloquear el flujo del programa principal, lo que mejora la capacidad de respuesta y la eficiencia.
- SPIFFS: SPIFFS (SPI Flash File System) es un sistema de archivos para dispositivos con memoria flash SPI. Esta librería permite leer y escribir archivos en la memoria flash del dispositivo, similar a cómo se haría en un sistema de archivos en una computadora. Es útil para almacenar configuraciones, archivos de datos, páginas web y otros recursos.
- AsyncTCP: Esta librería es una implementación asincrónica del protocolo TCP para los microcontroladores ESP32. Permite establecer conexiones TCP no bloqueantes, lo que mejora la eficiencia y capacidad de respuesta del dispositivo al manejar múltiples conexiones simultáneamente.
- freertos/FreeRTOS: Esta es la cabecera principal de FreeRTOS, un sistema operativo de tiempo real (RTOS) diseñado para sistemas embebidos. Incluye las definiciones y funciones básicas para crear y manejar tareas, colas,

temporizadores y otros recursos del sistema operativo. FreeRTOS permite ejecutar múltiples tareas concurrentemente y manejar los recursos del sistema de manera eficiente.

- freertos/task: Esta cabecera específica de FreeRTOS proporciona las funciones necesarias para crear, manejar y eliminar tareas. Las tareas en FreeRTOS son funciones que se ejecutan de manera concurrente.

- freertos/semphr: Esta cabecera proporciona las funciones necesarias para crear y manejar semáforos y mutexes (mutual exclusion). Los semáforos son utilizados para la sincronización entre tareas y/o interrupciones, mientras que los mutexes son usados para proteger secciones críticas y prevenir condiciones de carrera.

Capítulo 2

Desarrollo y Ejecución del Proyecto

En este capítulo se detallarán los pasos realizados para el desarrollo del proyecto. En la primera sección, se abordará el proceso de diseño de la PCB, que incluye el esquema del circuito, las capas y su estructura, así como las características específicas de la PCB resultante. Posteriormente, en la siguiente sección se explicará cómo se desarrolló la página web, abordando los archivos HTML, CSS y JavaScript. Luego, se detallará como se gestiona la página web y como se realiza la comunicación entre el ESP32 y la pagina. Además, se detalla como se procesan los datos enviados desde la pagina web a EPS32. Finalmente, en la ultima sección se describirá el código empleado para el control de los motores.

Diseño de la Placa de Circuito Impreso (PCB)

Esquemático del circuito

El esquemático del circuito se lo puede observar en la figura 2.1. El circuito esta compuesto por un ESP32 (Heltec WiFi Kit 32 v2), 5 borneras de dos pines y un LM35. 4 de las bornearas serán usadas para el control de los motores a paso mediante los drivers, mientras que una es para tierra y 5V. Por otro lado, el LM35 es para medir la temperatura del espacio donde se encuentra toda la parte de control del sistema.

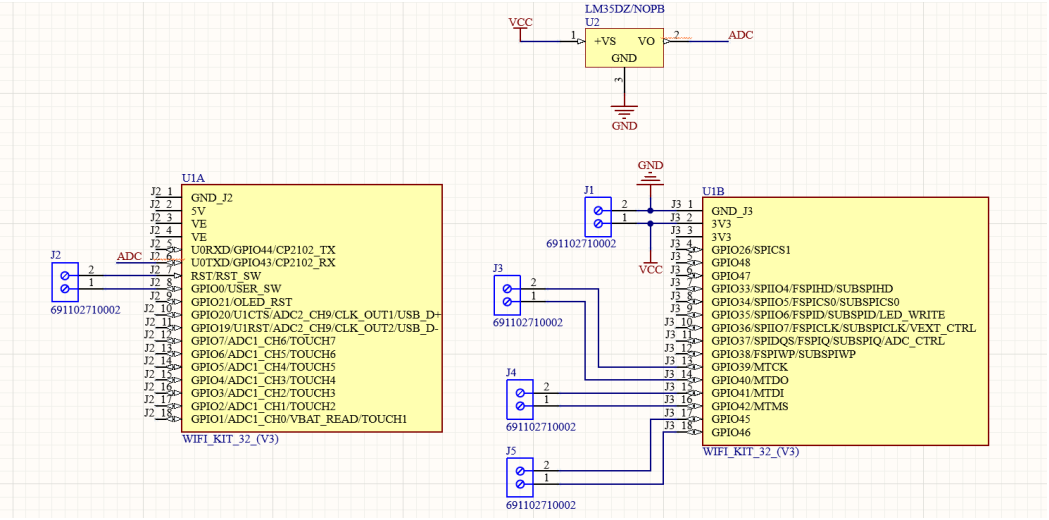


Figura 2.1: Esquemático del circuito.

Descripción de la PCB y sus respectivas características

La PCB cuenta con un tamaño de 54.86X74.3 mm y posee una densidad de 22.8 %. De la misma manera, esta posee 4 capas y un polygon pour conectado a tierra, La PCB resultante se muestra en la figura 2.2. La PCB cuenta con 4 capas: Top Layer, Layer 1, Layer 2, Bottom Layer.

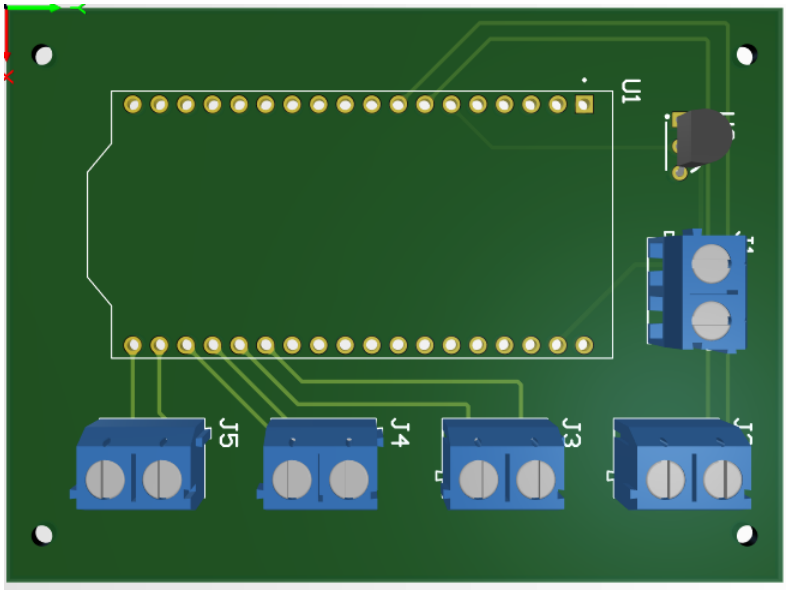


Figura 2.2: Diseño PCB 3D.

Ancho de Pista y otras características

El ancho mínimo para las rutas es de 0.13m, el ancho máximo de 0.508 mm y el ancho preferido de 0.254 mm como se muestra en la figura 2.3. Esos valores se establecieron siguiendo normas IPC. La PCB también contiene teardrops los cuales mejoran la integridad mecánica y eléctrica al reforzar la unión entre las pistas y los pads, reduciendo la probabilidad de roturas y fallos por fatiga del material. Con respecto al polygon pour, este ayudará a reducir el ruido eléctrico, mejorar la disipación de calor y proporcionar un retorno de corriente de baja impedancia, mejorando la estabilidad y el rendimiento del circuito. Finalmente los pads son de 2mm cada uno y gracias a ellos se podrá ensamblar la PCB en la apartado de control del sistema.

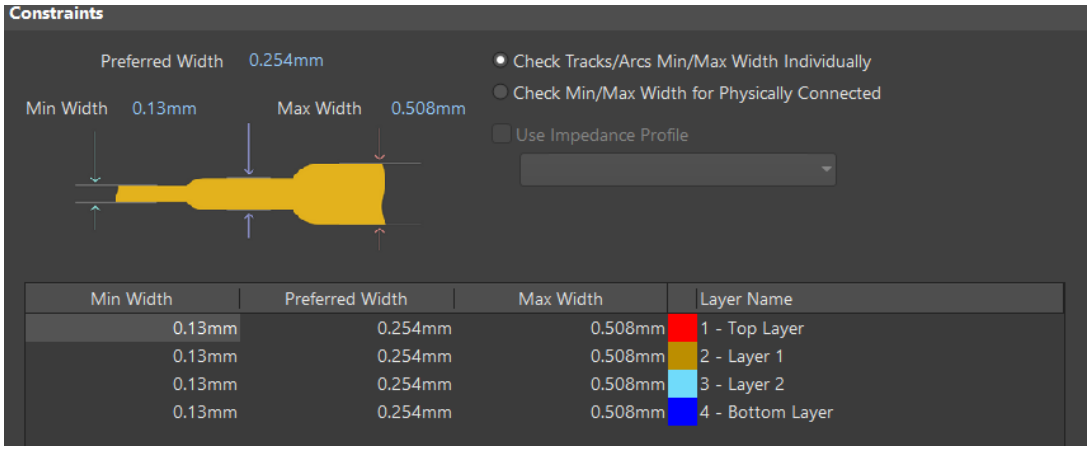


Figura 2.3: Ancho de Rutas.

Un problema que se encontró fue que la bornera de alimentación bloqueaba la entrada serial. De la misma manera, se noto que es necesario una bornera de 3.3 V para activar los motores. Por ende, se realizo otro diseño. Este se puede observar en la figura 2.4.

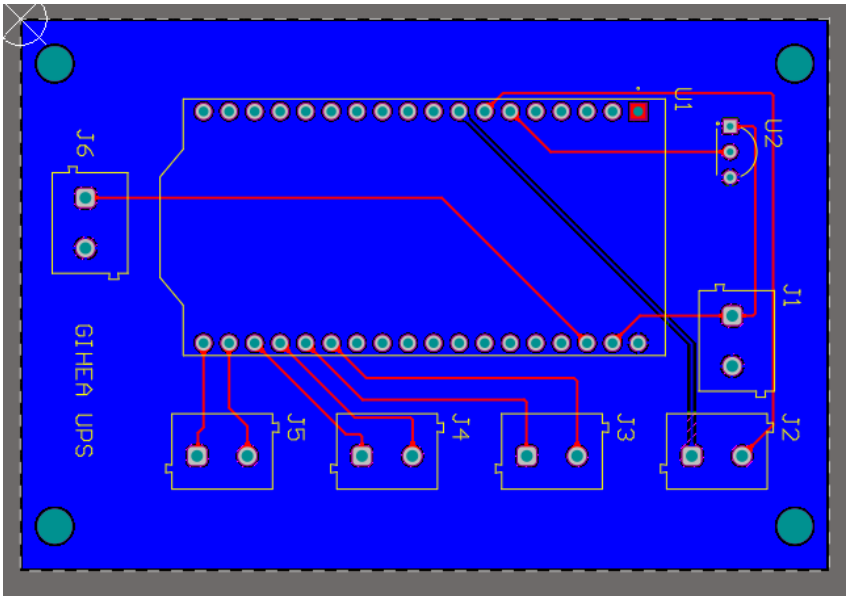


Figura 2.4: Diseño PCB Final.

De la misma manera, se cambio de 4 a 2 capas dado que se están usando pocas pistas.

Desarrollo de la página Web

Desarrollo de archivos HTML, CSS, y Javascript

Desarrollo archivo HTML

El archivo HTML contiene dos elementos principales: el head y el body. Dentro del head, primero se define el conjunto de caracteres a utilizar. Luego, se establecen los parámetros necesarios para que la página tenga un diseño responsivo. Después, se incluye el enlace de referencia al archivo CSS y, finalmente, se establece el título de la pestaña de la pagina. Esto se puede observar en la figura 2.5.

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="css/style.css" type="text/css">
    <title>Proyecto de Grado</title>
</head>
```

Figura 2.5: Head del archivo HTML.

Con respecto al body, este posee dos elementos principales: un header y un main. En el header se define el titulo de la pagina. Por otro lado, en el main se asignará el resto de contenido de la pagina. Esto se puede observar en la figura 2.6.

```
<header>
    <section id="til" class="title">
        <h1>Panel de Control de los Movimientos del Mecanismo</h1>
    </section>
</header>
```

Figura 2.6: Header del main.

Luego existe un elemento div, es en este elemento donde se definirá los bloques interactivos que conforman la pagina. Uno de estos de estos se puede observar en la figura 2.7. Se explicará solo uno de estos dado que son similares (Todo el archivo se lo puede encontrar en anexos).

```
<div class="est">
    <h1 class="names">Flexión/Extensión</h1>
    <figure id="fig2">
        <img  class="image" id="im1" data-image-number="1" alt="Fle/exte" src="Images/I3.png">
    </figure>

    <div class="inp">
        <label for="cord1">Posición:</label>
        <input type="number" id="cord1" name="cord1"  placeholder="Valores de 0-100">
        <button data-button-number="1"  class="buttons" id="B1" class="but" disabled>Accionar</button>
    </div>
</div>
```

Figura 2.7: Bloque interactivo.

Primero, se define un título con el texto "Flexión/Extensión". Luego, se inserta una figura, en este caso, la imagen Ï3". A continuación, se crea otro div que contiene una etiqueta "Posición:z un campo de entrada donde se puede introducir un valor numérico. Además, se define un botón que estará inicialmente deshabilitado. La idea es que, al hacer clic en la figura, el botón se habilite para enviar los datos. Por lo tanto, se debe hacer clic en la figura, introducir un valor en el campo de entrada y luego presionar el botón para que los datos sean enviados al ESP32.

Desarrollo archivo CSS

Con respecto al archivo CSS, aquí es donde se dará estilo a la página. Para hacer esto se hará uso de elementos, clases, o ids establecidos en el archivo HTML. En la figura 2.8 se muestran las características que tendrán los bloques interactivos pertenecientes al div con clase ïmágenes"(Todo el archivo se lo puede encontrar en anexos).

```css
img {
  display: block;
  margin: 0 auto;
  width: 25vw;
  height: 25vw;
  object-fit: contain;
  padding: 3vw;
  border: 2px solid black;
  border-radius: 0.5vw;
}
.image:hover {
  background-color: red;box-shadow: 0 0.1rem 5rem 0 rgba(0, 0, 0, 0.2);
}
.image.clicked {
  background-color: #8B0000;
}
.est {
  width: 50%;
  margin: 0 auto;
  text-align: center;
  display: flex; flex-direction: column;
}
.inp{
  margin: 1vw;
}
input{
  border-radius: 0.2vw; background-color: #FADADD;
}
label{
  color: #FADADD;
}
.names{
  padding-bottom: 1vw;
}
button{
  width: 4vw;background-color: #FADADD;  color: black;
}
```

Figura 2.8: Parámetros CSS para los bloques interactivos pertenecientes al div con clase ïmágenes".

Desarrollo archivo Javascript

Parte Uno: En la figura 2.9 se puede observar la primera parte del código.

```javascript
var targetUrl = `ws://${window.location.hostname}/ws`;
var websocket;
window.addEventListener('load', onLoad);

function onLoad() {
  initializeSocket();
}

function initializeSocket() {
  console.log('Opening WebSocket connection to ESP32...');
  websocket = new WebSocket(targetUrl);
  websocket.onopen = onOpen;
  websocket.onclose = onClose;
  websocket.onmessage = onMessage;
}
function onOpen(event) {
  console.log('Starting connection to server..');
}
function onClose(event) {
  console.log('Closing connection to server..');
  setTimeout(initializeSocket, 2000);
}
function onMessage(event) {
  console.log("WebSocket message received:", event)
}

function sendMessage(message) {
  websocket.send(message);
}
```

Figura 2.9: Código Javascript parte 1.

La variable target Url construye la URL del WebSocket utilizando el protocolo ws (WebSocket) y el hostname del sitio web donde se está ejecutando el script JavaScript en el navegador. Se asume que el servidor WebSocket está escuchando en el endpoint /ws. Con respecto a la variable websocket, esta se usará para mantener la instancia del objeto WebSocket. Después, se añade un event listener para 'load' que llama a la función onLoad. Cuando la ventana del navegador se carga por completo, onLoad a su vez invoca initializeSocket(), facilitando así el establecimiento de la conexión WebSocket.

La función initializeSocket() crea una nueva instancia de WebSocket utilizando la URL definida en targetUrl. También asigna funciones de callback para manejar eventos como la apertura de la conexión (onOpen), el cierre de la conexión (onClose),

y la recepción de mensajes (onMessage). La función onOpen(event) se ejecuta cuando la conexión WebSocket se abre correctamente. En este caso, simplemente muestra un mensaje en la consola. La función onClose(event) gestiona el cierre inesperado de la conexión, mostrando un mensaje de cierre en la consola y programando una reconexión después de un breve período. Mientras tanto, la función onMessage(event) maneja la recepción de mensajes desde el servidor WebSocket, mostrando los mensajes recibidos en la consola del navegador. Finalmente, la función sendMessage(message) permite enviar mensajes al servidor a través de la conexión establecida.

Parte Dos: En las figuras 2.10 y 2.11 se muestra la segunda parte del código.

```
document.addEventListener('DOMContentLoaded', function () {
    const images = document.querySelectorAll('.image');
    let prevClickedImage = null;

    images.forEach(function (image) {
        image.addEventListener('click', function () {
            if (prevClickedImage === this) {

                this.classList.toggle('clicked');
                prevClickedImage = null;
                disableButton(this);

            } else if (prevClickedImage === null) {

                this.classList.toggle('clicked');
                prevClickedImage = this;
                enableButton(this);
            }
        });
    });
```

Figura 2.10: Código Javascript parte 2.1.

El codigo de la figura 2.10 contiene una función que chequea si una imagen se le ha hecho clic. Si a ninguna imagen se le ha hecho click entonces se puede hacer clic a cualquier imagen. Una vez hecho click a una imagen, se le da a esta imagen la clase'Clicked' y se habilita el botón para enviar datos. Si ya se ha hecho clic a una imagen, entonces solo se puede hacer clic a esa imagen para quitarle la clase 'Clicked' y deshabilitar el botón.

```javascript
function disableButton(image) {
    const buttonId = 'B' + image.dataset.imageNumber;
    const button = document.getElementById(buttonId);
    if (button) {
        button.disabled = true;
    } else {
        console.error("Button element not found with ID: " + buttonId);
    }
}

function enableButton(image) {
    const buttonId = 'B' + image.dataset.imageNumber;
    const button = document.getElementById(buttonId);
    if (button) {
        button.disabled = false;
    } else {
        console.error("Button element not found with ID: " + buttonId);
    }
}
```

Figura 2.11: Código Javascript parte 2.2.

El código de la figura 2.11 contiene dos funciones, disableButton y enableButton, que gestionan el estado de habilitación de un botón HTML. Ambas funciones reciben como parámetro un elemento de imagen (image). Utilizan el atributo dataset.imageNumber de la imagen para construir el ID del botón correspondiente (ej. 'B1', 'B2'). Luego, buscan el botón en el documento mediante getElementById utilizando ese ID. Si encuentran el botón, disableButton lo deshabilita (button.disabled = true), mientras que enableButton lo habilita (button.disabled = false). Si no encuentran el botón, registran un mensaje de error en la consola indicando que no se encontró un elemento con el ID especificado.

Parte Tres: En las figuras 2.11 y 2.12 se muestra la tercera parte del código.

```javascript
const buttons = document.querySelectorAll('.buttons');

buttons.forEach(function (button) {
    button.addEventListener('click', function () {
        const buttonNumber = event.target.dataset.buttonNumber;
        sendData(buttonNumber);
    });
});


function sendData(buttonNumber) {
    // Get the numeric value entered by the user
    const numericInput = parseFloat(document.getElementById('cord' + buttonNumber).value);

    // Create a data object to send
    const data = {
        image: 'Image' + buttonNumber,
        numericValue: numericInput
    };
    const dataString = `image=${data.image} numericValue=${data.numericValue}`;

    sendMessage(dataString);

    resetAll();
}
```

Figura 2.12: Código Javascript parte 3.1.

El código de la figura 2.12 define una funcionalidad para gestionar clics en una colección de botones HTML con la clase 'buttons'. Primero, selecciona todos los botones usando document.querySelectorAll('.buttons') y luego añade un escuchador de eventos de clic a cada uno. Cuando se hace clic en un botón, se obtiene el número del botón desde su atributo data-button-number y se pasa a la función sendData. Esta función toma el número del botón, obtiene el valor numérico de un campo de entrada asociado (cord seguido del número del botón), y crea un objeto de datos con la imagen correspondiente y el valor numérico. Luego, convierte estos datos en una cadena de texto y llama a sendMessage para enviarlos, seguido de una llamada a resetAll para restablecer el estado.

```
    // Function to reset everything
    function resetAll() {
        // Clear input fields
        document.querySelectorAll('input').forEach(function(input) {
            input.value = '';
          });

        // Reset clicked images
        document.querySelectorAll('.image.clicked').forEach(function (image) {
            image.classList.remove('clicked');
        });

        // Disable all buttons
        document.querySelectorAll('button').forEach(function (button) {
            button.disabled = true;
        });

        prevClickedImage = null;
    }
});
```

Figura 2.13: Código Javascript parte 3.2.

El código de la figura 2.13 define una función resetAll que restablece el estado de ciertos elementos en la página web. Primero, limpia todos los campos de entrada (input) estableciendo sus valores a una cadena vacía. Luego, elimina la clase 'clicked' de todas las imágenes que tengan esta clase (.image.clicked). A continuación, deshabilita todos los botones estableciendo su propiedad disabled a true. Finalmente, establece la variable prevClickedImage a null. Esta función se asegura de que todos los campos de entrada, imágenes y botones vuelvan a su estado inicial, permitiendo un nuevo inicio limpio en la interfaz de usuario.

Comunicación con el ESP32 y Manejo de Pagina web

Esta sección esta compuesta de tres partes: Wifi, Servidor, y Procesamiento de Datos.

Wifi

Para poder comunicar la pagina web con el ESP32, primero se tiene que conectar el microcontrolador a una red Wifi. Para esto se configuro el código presente

en las figuras 2.14 y 2.15.

```
#include <WiFi.h>
#include "conexion.h"

void wifi()
{
  const char *ssid = "Xtrim_JBG-2G";
  const char *password = "estuardo61";

  IPAddress local_IP(192, 168, 1, 17);
  IPAddress gateway(192, 168, 1, 1);
  IPAddress subnet(255, 255, 255, 0);
  if (!WiFi.config(local_IP, gateway, subnet)) {
    Serial.println("STA Failed to configure");
  }
  Serial.println("Scanning availabe networks.....");
  int n = WiFi.scanNetworks();
  if (n != 0)
  {
    Serial.print(n);
    Serial.println(" network(s) found");
    for (int i = 0; i < n; i++)
    {
      Serial.print("network ");
      Serial.print(i + 1);
      Serial.print(": ");
      Serial.print(WiFi.SSID(i));
      Serial.print(" (");
      Serial.print(WiFi.RSSI(i));
      Serial.print(")");
      Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN) ? " open " : " ***");
      delay(50);
    }
  }
  else
  {
    Serial.println("no available networks found");
  }
```

Figura 2.14: Código Wifi parte 1.

Primero lo que se hizo fue incluir dos header, el de WiFi y el de conexion. El header de WiFi permite usar las funciones de dicha librería mientras que el header conexión es usado para declarar funciones; en este caso la función wifi. Luego, se define el nombre de la red y su contraseña. Después, se establece la ip, gateway y subred. Una vez hecho esto, se configuran estos parámetros para la red. Luego, para confirmar que la red existe, se manda a imprimir todas las redes disponibles con su numero y tipo de encriptación .

```
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());

  if (!SPIFFS.begin(true))
  {
    Serial.println("Failed to mount SPIFFS");
    return;
  }
  Serial.println("SPIFFS mounted successfully");
}
```

Figura 2.15: Código Wifi parte 2.

Con respecto al código de la figura 2.15, primero imprime el SSID de la red WiFi. Luego, usa WiFi.begin(ssid, password) para iniciar la conexión a la red WiFi. Mientras la conexión no se establezca, el dispositivo imprime puntos en el monitor serie cada 500 milisegundos para indicar que sigue intentando conectarse. Una vez conectado, imprime un mensaje confirmando la conexión y muestra la dirección IP local obtenida (WiFi.localIP()). Después, intenta montar el sistema de archivos SPIFFS con SPIFFS.begin(true). Si falla, imprime un mensaje de error y finaliza. Si tiene éxito, imprime un mensaje de éxito.

Servidor

La pagina web se guarda en la flash del ESP32, por ende se tiene que realizar las configuraciones presentes en la figura 2.16.

```
AsyncWebServer server(80);
AsyncWebSocket ws("/ws");

void web()
{
  ws.onEvent(onWsEvent);
  server.addHandler(&ws);

  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request)
            {
              Serial.printf("Requesting index page...\n");
              request->send(SPIFFS, "/main.html", "text/html",false); });

  server.on("/css/style.css", HTTP_GET, [](AsyncWebServerRequest *request)
            {
              Serial.printf("Requesting style css...\n");
              request->send(SPIFFS, "/css/style.css", "text/css"); });

  server.on("/script/J1.js", HTTP_GET, [](AsyncWebServerRequest *request)
            {
              Serial.printf("Requesting socket js...\n");
              request->send(SPIFFS, "/script/J1.js", "text/javascript"); });

  server.on("/Images/I1.png", HTTP_GET, [](AsyncWebServerRequest *request)
            { request->send(SPIFFS, "/Images/I1.png", "image/png"); });

  server.on("/Images/I2.png", HTTP_GET, [](AsyncWebServerRequest *request)
            { request->send(SPIFFS, "/Images/I2.png", "image/png"); });

  server.on("/Images/I3.png", HTTP_GET, [](AsyncWebServerRequest *request)
            { request->send(SPIFFS, "/Images/I3.png", "image/png"); });
  server.onNotFound(notFound);
  server.begin();
}
```

Figura 2.16: Envió de archivos pagina web.

Primero se inicializa un servidor web asíncrono y un servidor de WebSocket asíncrono. Luego, se define la función web que establece un controlador de eventos para el WebSocket y lo añade al servidor web. Luego, se configura varias rutas para manejar solicitudes HTTP GET, sirviendo archivos HTML, CSS, JavaScript y sus respectivas imágenes desde el sistema de archivos SPIFFS. Además, se establece un controlador para manejar solicitudes no encontradas y se inicia el servidor web para que comience a aceptar solicitudes. La función también incluye mensajes en el monitor serie para indicar cuándo se solicitan y envían los diferentes recursos.

```
void onWsEvent(AsyncWebSocket *server, AsyncWebSocketClient *client, AwsEventType type, void *arg, uint8_t *data, size_t len)
{
  switch (type)
  {
  case WS_EVT_CONNECT:
  {
    Serial.printf("WS event connect\n"); break;
  }
  case WS_EVT_DISCONNECT:
  {
    Serial.printf("WS event disconnect\n"); break;
  }
  case WS_EVT_DATA:
  {
    AwsFrameInfo *info = (AwsFrameInfo *)arg;
    if (info->final && info->index == 0 && info->len == len)
    {
      if (info->opcode == WS_TEXT)
      {
        data[len] = 0;
        char *command = (char *)data;
        movement(command);
      }
    }
    break;
  }
  case WS_EVT_PONG:
  {
    break;
  }
  case WS_EVT_ERROR:
  {
    break;
  }
  }
}
```

Figura 2.17: Gestión de Eventos WebSocket Asíncronos

De la misma manera, en la figura 2.17 se muestra como se gestionan los eventos de la pagina. La función onWsEvent maneja eventos de WebSocket para el servidor, recibiendo parámetros como el servidor, el cliente, el tipo de evento, datos y su longitud. Utiliza un switch para gestionar distintos tipos de eventos: al conectar, se imprime un mensaje indicando la conexión; al desconectar, imprime otro mensaje indicando su desconexión. Por otra parte, cuando recibe datos, verifica si el marco de datos es final, el índice es 0 y la longitud es correcta. Si los datos son texto, los convierte en una cadena y llama a la función movement con el comando recibido. En otras palabras, la variable command contenderá el valor numérico y el numero de imagen enviado desde la pagina web.

Procesamiento de Datos

```
void movement(const char *command)
{
  std::string commands(command);

  size_t imagePos = commands.find("image=");
  size_t spaceAfterImage = commands.find(" ", imagePos);
  size_t numericValuePos = commands.find("numericValue=");

  if (imagePos != std::string::npos && spaceAfterImage != std::string::npos && numericValuePos != std::string::npos)
  {

    std::string image = commands.substr(imagePos + 6, spaceAfterImage - (imagePos + 6));
    std::string numericValue = commands.substr(numericValuePos + 13);

    std::cout << "Numeric Value: " << numericValue << std::endl;
    ejecucion(image, numericValue);
  }
}

void ejecucion(std::string tipo, std::string distancia)
{
  if (tipo == "Image1")
  {
    std::cout << "Image: " << tipo << std::endl;
  }
  else if (tipo == "Image2")
  {
    std::cout << "Image: " << tipo << std::endl;
  }
  else if (tipo == "Image3")
  {
    std::cout << "Image: " << tipo << std::endl;
  }
}
```

Figura 2.18: Procesamiento de datos.

La función movement de la figura 2.18, recibe los datos de la pagina web y los convierte en un objeto std::string para facilitar su manipulación. Luego, busca las subcadenas image= y numericValue= dentro de este string. Si encuentra ambas subcadenas junto con el espacio después de ïmage=", extrae el nombre de la imagen y el valor numérico asociado. Posteriormente, imprime el valor numérico y llama a la función ejecucion con estos valores para llevar a cabo acciones específicas dependiendo del tipo de imagen y valor numérico proporcionados. La función ejecucion, por otro lado, recibe dos parámetros: tipo (indicando el tipo de imagen) y distancia (el valor numérico asociado). Basado en el tipo de imagen recibido (Image1, Image2, Image3), realiza acciones específicas. Esta función está diseñada para gestionar diferentes tipos de imágenes y ejecutar tareas correspondientes según los parámetros recibidos. en la siguiente sección se detallara que acciones se realizaran segun el tipo de imagen.

Control de motores

Una vez realizado la comunicación, se procede a activar los motores dependiendo que movimiento se activo. Para realizar esto, primero se deben inicializar los motores Lo primero que se debe realizar es establecer 2 pines para cada motor, donde *dir* representara la dirección, mientras que *step* el numero de pasos. Esto se puede observar en la figura 2.19.

```
//--------------Motor1-----------------------------
#define dirM1 21
#define stepM1 22

//--------------Motor2-----------------------------
#define dirM2 19
#define stepM2 23

//--------------Motor3-----------------------------
#define dirM3 18
#define stepM3 5

//--------------Motor4-----------------------------
#define dirM4 27
#define stepM4 14
```

Figura 2.19: Pines para el Control Motores.

Ahora, se debe configurar cada motor. Primero, se crean cuatro objetos AccelStepper (M1, M2, M3, M4) para controlar cuatro motores paso a paso. El primer parámetro 1 indica el tipo de controlador (1 = DRIVER), y stepM1, dirM1, etc., son los pines de paso y dirección para cada motor. Ver figura 2.20.

```
AccelStepper M1 = AccelStepper(1, stepM1, dirM1);
AccelStepper M2 = AccelStepper(1, stepM2, dirM2);
AccelStepper M3 = AccelStepper(1, stepM3, dirM3);
AccelStepper M4 = AccelStepper(1, stepM4, dirM4);
```

Figura 2.20: Definición de Objetos AccelStepper.

Luego, se declaran los manejadores de tareas (TaskHandle_t) y los semáforos (SemaphoreHandle_t) para cada motor. Ver figura 2.21.

```
TaskHandle_t mov1TaskHandle = NULL;
TaskHandle_t mov2TaskHandle = NULL;
TaskHandle_t mov3TaskHandle = NULL;
TaskHandle_t mov4TaskHandle = NULL;

SemaphoreHandle_t mov1Semaphore = NULL;
SemaphoreHandle_t mov2Semaphore = NULL;
SemaphoreHandle_t mov3Semaphore = NULL;
SemaphoreHandle_t mov4Semaphore = NULL;
```

Figura 2.21: Manejo de Tareas y Semáforos.

La función initTasks crea un semáforo binario y una tarea para cada motor. xTaskCreate se usa para crear una tarea en FreeRTOS. Cada tarea tiene un tamaño de pila de 2048 bytes y una prioridad de 1. Ver figura 2.22.

```
void initTasks()
{
    mov1Semaphore = xSemaphoreCreateBinary();
    xTaskCreate(Motor1, "Motor1", 2048, NULL, 1, &mov1TaskHandle);

    mov2Semaphore = xSemaphoreCreateBinary();
    xTaskCreate(Motor2, "Motor2", 2048, NULL, 1, &mov2TaskHandle);

    mov3Semaphore = xSemaphoreCreateBinary();
    xTaskCreate(Motor3, "Motor3", 2048, NULL, 1, &mov3TaskHandle);

    mov4Semaphore = xSemaphoreCreateBinary();
    xTaskCreate(Motor4, "Motor4", 2048, NULL, 1, &mov4TaskHandle);
}
```

Figura 2.22: Inicialización de Tareas y Semáforos

La función initF establece la velocidad máxima y la aceleración para cada motor. Ver figura 2.23.

```
void initTasks()
{
    mov1Semaphore = xSemaphoreCreateBinary();
    xTaskCreate(Motor1, "Motor1", 2048, NULL, 1, &mov1TaskHandle);

    mov2Semaphore = xSemaphoreCreateBinary();
    xTaskCreate(Motor2, "Motor2", 2048, NULL, 1, &mov2TaskHandle);

    mov3Semaphore = xSemaphoreCreateBinary();
    xTaskCreate(Motor3, "Motor3", 2048, NULL, 1, &mov3TaskHandle);

    mov4Semaphore = xSemaphoreCreateBinary();
    xTaskCreate(Motor4, "Motor4", 2048, NULL, 1, &mov4TaskHandle);
}
```

Figura 2.23: Inicialización de Parámetros de los Motores

Las funciones (Motor1, Motor2, Motor3, Motor4) son las tareas que se ejecutan en FreeRTOS para controlar cada motor. mov1TaskHandle = NULL; y vTaskDelete(NULL); aseguran que una vez que Motorx() ha terminado su ejecución, la tarea se elimina adecuadamente y se liberan los recursos asociados. Además, establecer el manejador de tarea a NULL ayuda a mantener el estado del sistema claro y preparado para la posible recreación de la tarea, mientras que vTaskDelete(NULL); asegura que la tarea no siga ejecutándose ni consuma recursos innecesarios. Ver figura 2.24.

```
void Motor1(void *pvParameters)
{
    const int yieldInterval = 1000;
    printf("The value of myInt is: %d\n", steps);
    int stepsTaken;


    if (xSemaphoreTake(mov1Semaphore, portMAX_DELAY) == pdTRUE)
    {
        M1.moveTo(steps);
        stepsTaken = 0;


        while (M1.distanceToGo() != 0)
        {
            M1.run();
            stepsTaken++;


            if (stepsTaken >= yieldInterval)
            {
                stepsTaken = 0;
                vTaskDelay(pdMS_TO_TICKS(1));
            }
        }
    }

    steps = 0;

    mov1TaskHandle = NULL;
    vTaskDelete(NULL);
}
```

Figura 2.24: Funciones de Control de Motores

La función TMotor1() asegura que la tarea Motor1 esté activa y que el semáforo asociado esté disponible para su uso. Primero, libera el semáforo mov1Semaphore si ha sido creado, permitiendo que la tarea Motor1, que podría estar esperando para comenzar, pueda proceder. Esto es crucial para que Motor1 pueda acceder a los recursos necesarios y ejecutar su código correctamente, garantizando que el motor 1 pueda moverse según las instrucciones dadas. Esto se realizará para todos los motores. Ver figura 2.25.

```
void TMotor1()
{
  if (mov1Semaphore != NULL)
  {
    xSemaphoreGive(mov1Semaphore);
  }

  if (mov1TaskHandle == NULL)
  {
    xTaskCreate(Motor1, "Motor1", 2048, NULL, 1, &mov1TaskHandle);
  }
}
```

Figura 2.25: Inicio o Reactivación la Tarea del Motor 1 en FreeRTOS

De la misma manera, TMotor1() verifica si la tarea Motor1 ya ha sido creada al comprobar si mov1TaskHandle es NULL. Si la tarea no existe, la función la crea usando xTaskCreate. Esto se hace para asegurar que Motor1 esté en ejecución y pueda gestionar el movimiento del motor 1 de manera eficiente en un entorno multitarea. Este proceso permite coordinar la ejecución de múltiples tareas en FreeRTOS, optimizando el uso de los recursos y asegurando una operación sincronizada y eficiente de los motores en el sistema.

```
void ejecucion(std::string tipo, std::string distancia)
{
  if (tipo == "Image1")
  {
    std::cout << "Image: " << tipo << std::endl;
    steps = std::stoi(distancia);
    TMotor1();
  }
  else if (tipo == "Image2")
  {
    std::cout << "Image: " << tipo << std::endl;
  }
  else if (tipo == "Image3")
  {
    std::cout << "Image: " << tipo << std::endl;
  }
}
```

Figura 2.26: Coordinación y Ejecución

Finalmente, en la función *ejecucion* se tendrá que coordinar cuando se activa cada motor y cuantos pasos se realizarán. La variable *steps* es una variable global que esta relacionada con el valor numérico enviado desde la interfaz. Esta se usará en las funciones Motorx. Ver figura 2.26.

Capítulo 3

Pruebas y Análisis de Resultados

En este capítulo se detallan las pruebas realizadas, en la PCB, página web, y los movimientos del efector final del sistema, fundamentales para garantizar el correcto funcionamiento e integración del sistema completo. La PCB, una placa de cuatro capas, controla los motores a pasos que facilitan los movimientos del sistema. Se presentan imágenes de la PCB fabricada y ensamblada, y se reportan los exitosos resultados de las pruebas de continuidad de componentes clave como el ESP32, las borneras y el sensor LM35. Además, se describen las pruebas de la página web, que incluyen la conexión a la red WiFi, el acceso a la interfaz web y la verificación de la transmisión de datos hacia el sistema, asegurando una comunicación eficiente. Finalmente, se discuten las pruebas realizadas para el efector final pueda ser capaz de realizar los tres movimientos de muñeca.

Pruebas PCB

En la figura 2.19 se puede observar la PCB ya fabricada. Como se expuso en el capítulo dos, esta PCB tiene 4 capas y su función es controlar los cuatro motores a pasos responsables de los movimientos del sistema.

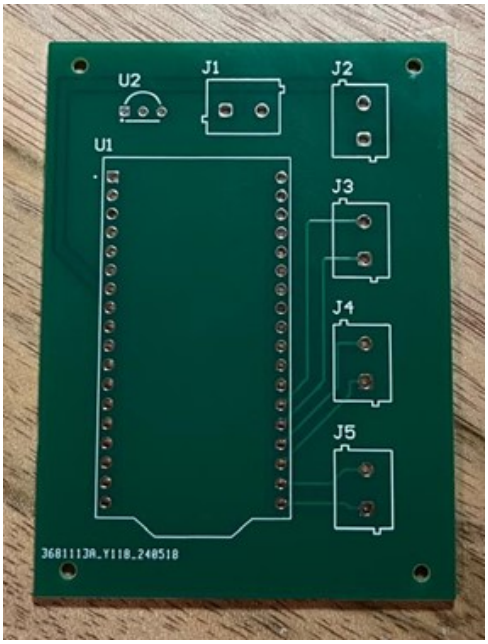


Figura 3.1: PCB Física.

En la figura 2.20 se puede observar la PCB ya ensamblada con sus respectivos componentes.

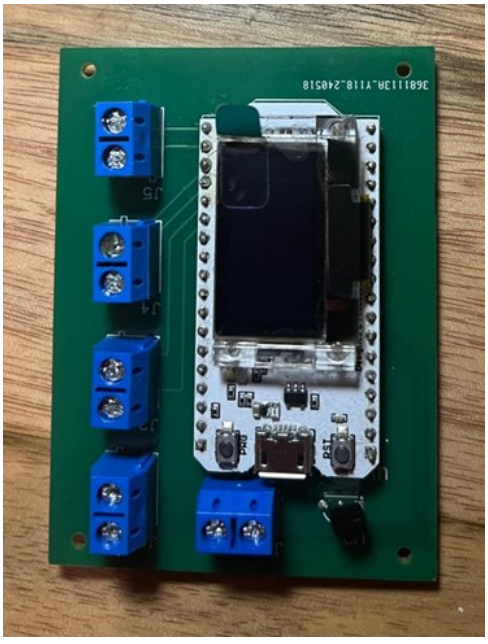


Figura 3.2: PCB Física con sus respectivos componentes.

Se realizó pruebas de continuidad del ESP32, las borneras y el LM35 y estas resultaron un éxito.

Pruebas Pagina Web

Para realizar las pruebas de la pagina web, primero se tiene que conectar a la red Wifi. Esto se muestra en la figura 2.21.

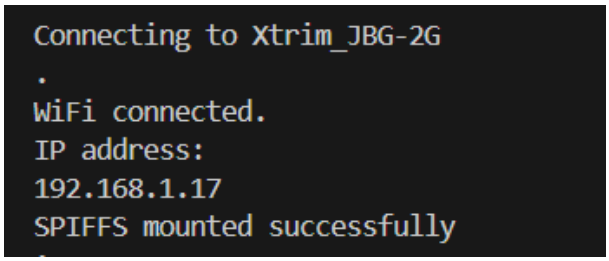


Figura 3.3: Conexión red wifi.

En la figura 2.21, se puede notar que el ESP32 se a logrado conectar a la red wifi. De la misma manera, su ip es el 192.168.1.17 como se estableció a un inicio y el SPIFFS se ha montado con éxito. Ahora, para acceder a la pagina web, se tiene que acceder a la ip mediante un navegador. Si todo a sale bien debe aparecer los mensajes de la figura 2.22.

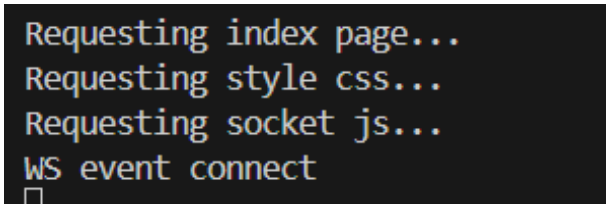


Figura 3.4: Eventos en la Carga de una Página Web.

La pagina web se puede observar en la figura 2.23.

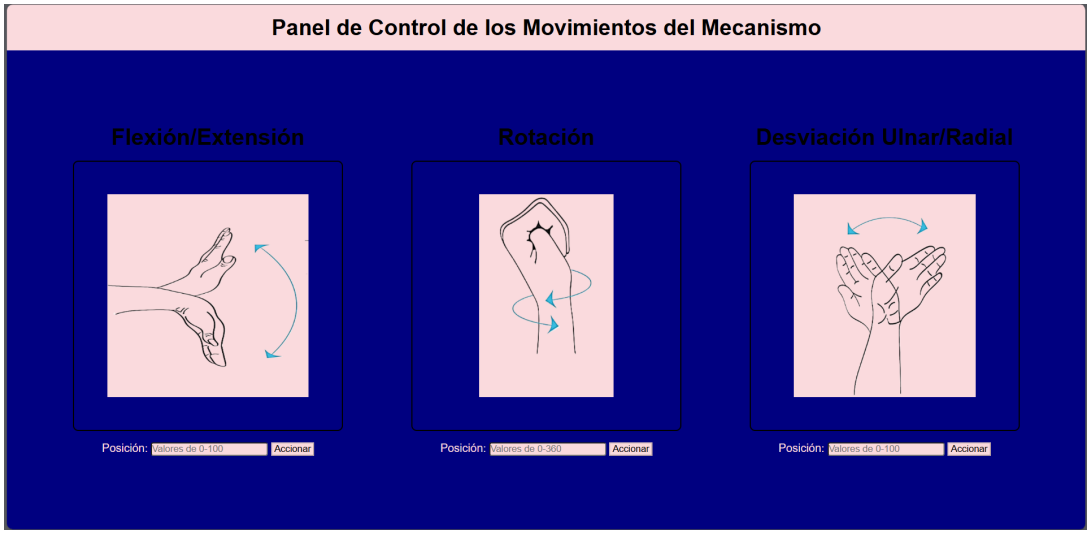


Figura 3.5: Pagina Web.

Ahora, para el envió de datos se tiene que seleccionar una opción (imagen) y escribir los datos en la entrada numérica. Una vez hecho esto, se pulsa el botón accionar y el valor numérico como el número de imagen tendrán que ser impresos en la entrada serial. El procedimiento y los resultados se pueden ver en la figura 2.24 y 2.25 respectivamente.

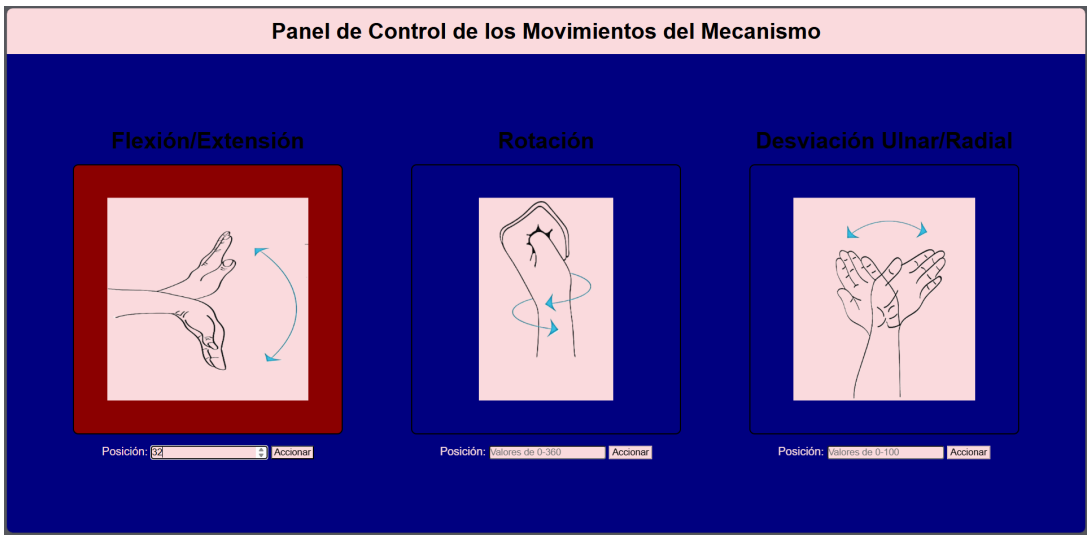


Figura 3.6: Procedimiento envió de datos.

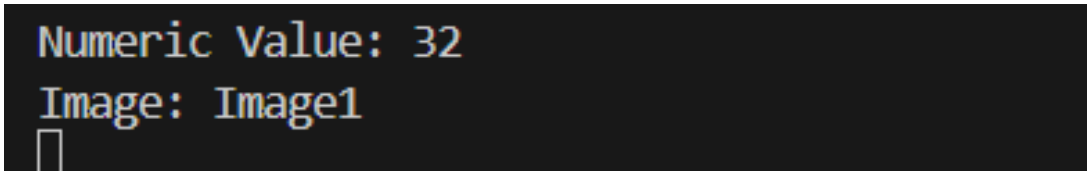


Figura 3.7: Resultados.

Una vez hecho esto, se pueden realizar el número de movimientos necesarios para cada opción.

Pruebas Movimientos de los Motores

Para estas pruebas, se envió desde la interfaz una valor que representa los pasos de motor. El eje del motor giro sin ningún inconveniente.

Capítulo 4

Conclusiones y Trabajos Futuros

En conclusión, se logró desarrollar la página web, el diseño de la PCB del sistema y se describieron las características y tecnologías empleadas en la prótesis de muñeca. La comunicación efectiva entre la interfaz y el ESP32 permitió un control preciso de los motores. De la misma manera, dado que este sistema hace uso de mecanismos paralelos, este sistema posee un menor riesgo de sufrir deformaciones y errores acumulativos. Con respecto a su implementación, la parte electrónica y de control se desarrollo con éxito, sin inconvenientes notables. No obstante, surgieron problemas en la parte mecánica, particularmente en la integración de los componentes y la estabilidad de la estructura. Estos problemas requieren atención especial para asegurar que el sistema se adapte adecuadamente a una prótesis de muñeca. Se recomienda una revisión exhaustiva de la estructura mecánica y la incorporación de mejoras en el diseño para superar estos desafíos. Como trabajo futuro, se planea la programación del sensor LM35 para medir la temperatura ambiente en el entorno de los componentes, lo que permitirá monitorear y garantizar el correcto funcionamiento del sistema en diferentes condiciones térmicas. Adicionalmente, se sugiere realizar pruebas de durabilidad y resistencia para validar la fiabilidad de los componenetes a lo largo del tiempo.

Glosario

ABENICS Mecanismo de rótula activa con tres grados de libertad basado en engranajes esféricos.

CS-gear Engranaje esférico cruzado.

DOF Grados de libertad.

MP-gear Engranaje monopolo.

Referencias

[1] N. Bajaj, A. Spiers y A. Dollar, «State of the Art in Artificial Wrists: A Review of Prosthetic and Robotic Wrist Design,» *IEEE Transactions on Robotics*, vol. 35, n.º 1, págs. 261-277, feb. de 2019, ISSN: 15523098. DOI: 10.1109/TRO.2018.2865890.

[2] A. Zinck, Ø. Stavdahl, E. Biden y P. Kyberd, «Design of a compact, reconfigurable, prosthetic wrist,» *Applied Bionics and Biomechanics*, vol. 9, n.º 1, págs. 117-124, 2012, ISSN: 17542103. DOI: 10.3233/ABB-2011-0043.

[3] S. Fan, S. Fan, L. Jiang y H. Liu, «A Design of a Miniaturized Prosthetic Wrist Based on Repetition Rate of Human Wrist Daily Tasks,» en *2016 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2016, págs. 1643-1648, ISBN: 978-1-5090-4364-4. DOI: 10.1109/ROBIO.2016.7866563.

[4] M. Johannes, J. Bigelow, J. Burck, S. Harshbarger, M. Kozlowski y T. Van Doren, «An Overview of the Developmental Process for the Modular Prosthetic Limb,» *Johns Hopkins APL Technical Digest*, vol. 30, n.º 3, págs. 207-216, 2011.

[5] C. Gosselin, E. St Pierre y M. Gagne, «On the Development of the Agile Eye,» *IEEE Robotics & Automation Magazine*, vol. 3, n.º 4, págs. 29-36, 1996, ISSN: 1558-223X. DOI: 10.1109/100.556480.

[6] H. Fan, G. Wei y L. Ren, «Prosthetic and robotic wrists comparing with the intelligently evolved human wrist: A review,» *Robotica*, vol. 40, n.º 11, págs. 4169-4191, nov. de 2022, ISSN: 14698668. DOI: 10.1017/S0263574722000856.

[7] B. Yuan, D. Hu, S. Gu, S. Xiao y F. Song, «The global burden of traumatic amputation in 204 countries and territories,» *Frontiers in Public Health*, vol. 11, pág. 10, 2023, ISSN: 22962565. DOI: 10.3389/fpubh.2023.1258853.

[8] R. Berger, «The anatomy and basic biomechanics of the wrist joint,» *Journal of Hand Therapy*, vol. 9, n.º 2, págs. 84-93, 1996, ISSN: 08941130. DOI: 10.1016/S0894-1130(96)80066-4.

[9] S. Divya, «Design of Wrist and Forearm Mechanisms for Enhanced Humanoid Dexterity,» Tesis doct., 2021, pág. 5.

[10] F. Thomas, «Kinematics of a Gear-Based Spherical Mechanism,» en *Advances in Robot Kinematics 2022*, 2022, págs. 323-331. DOI: 10.1007/978-3-031-08140-8{_}35.

[11] K. Abe, K. Tadakuma y R. Tadakuma, «ABENICS: Active Ball Joint Mechanism With Three-DoF Based on Spherical Gear Meshings,» *IEEE TRANSACTIONS ON ROBOTICS*, vol. 37, n.º 5, págs. 1806-1825, 2021, ISSN: 1941-0468. DOI: 10.1109/TRO.2021.3070124.

[12] Heltec Automation, *WiFi Kit 32 Development Kit*. dirección: https://resource.heltec.cn/download/WiFi_Kit_32/WiFi%20Kit32.pdf.

[13] bullkman3d, *TB6600 Stepper Motor Driver Safety Statement*. dirección: https://bulkman3d.com/wp-content/uploads/2019/06/TB6600-Stepper-Motor-Driver-BM3D-v1.1.pdf.

Anexos A

Código HTML

```html
<!DOCTYPE html>
<html lang="es">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <link rel="stylesheet" href="css/style.css" type="text/css">
    <title>Proyecto de Grado</title>
</head>

<body>
    <header>
        <section id="til" class="title">
            <h1>Panel de Control de los Movimientos del Mecanismo</h1>
        </section>
    </header>

    <main>

        <div class="images">

            <div class="est">
                <h1 class="names">Flexión/Extensión</h1>
                <figure id="fig2">
                    <img  class="image" id="im1" data-image-number="1" alt="Fle/exte" src="Images/I3.png">
                </figure>

                <div class="inp">
                    <label for="cord1">Posición:</label>
                    <input type="number" id="cord1" name="cord1"  placeholder="Valores de 0-100">
                    <button data-button-number="1"  class="buttons" id="B1" class="but" disabled>Accionar</button>
                </div>
            </div>
```

```
            <div class="est">
                <h1 class="names">Rotación</h1>
                <figure id="fig2">
                    <img class="image"  id="im2" data-image-number="2" alt="rotation" src="Images/I2.png">
                </figure>
                <div class="inp">
                    <label for="cord2">Posición:</label>
                    <input type="number" id="cord2"  name="cord2" placeholder="Valores de 0-360">
                    <button  data-button-number="2" class="buttons"  id="B2" class="but" disabled>Accionar</button>
                </div>

            </div>


            <div class="est">
                <h1 class="names">Desviación Ulnar/Radial</h1>
                <figure id="fig3">
                    <img  class="image" id="im3" data-image-number="3" alt="radi/ulna" src="Images/I1.png">
                </figure>
                <div class="inp">
                    <label for="cord3">Posición:</label>
                    <input type="number" id="cord3" name="cord3" placeholder="Valores de 0-100">
                    <button data-button-number="3"  class="buttons" id="B3" class="but" disabled>Accionar</button>
                </div>
            </div>

        </div>
    </main>

    <script src="script/J1.js"></script>
</body>
</html>
```

Anexos B

Código CSS

```css
*, ::after, ::before{
    box-sizing: border-box;
    padding: 0;
    margin: 0;
  }

  /*-------------------------body-----------------------------------*/
  body{
    font-family:arial, Verdana, Geneva, Tahoma, sans-serif;
  }

/*-------------------------------header--------------------------------*/

header{
    display: flex;
    justify-content: center;
    align-items: center;
    padding: 1vh;
    background-color: #FADADD;
}

.title{
    display: flex;
    justify-content: center;
    align-items: center;
    font-size: 1.3rem;
    padding: 1vh;
}


main{
  padding: 2vh;
  padding-top: 10vh;
  height: 91.3vh;
  background-color: #000080;
}
```

```css
.images{
  display: flex;
  margin: 2vw;
  justify-content: center;
  align-items: center;
}
/*-----------------------------------Imagenes-----------------------------------*/

img {
  display: block;
  margin: 0 auto;
  width: 25vw;
  height: 25vw;
  object-fit: contain;
  padding: 3vw;
  border: 2px solid black;
  border-radius: 0.5vw;
}

.image:hover {
  background-color: red;
  box-shadow: 0 0.1rem 5rem 0 rgba(0, 0, 0, 0.2);
}

.image.clicked {
  background-color: #8B0000;
}


.est {
  width: 50%;
  margin: 0 auto;
  text-align: center;
  display: flex;
  flex-direction: column;
}
```

```css
.inp{
  margin: 1vw;
}

input{
  border-radius: 0.2vw;
  background-color: #FADADD;
}

label{
  color: #FADADD;
}

.names{
  padding-bottom: 1vw;
}

button{
  width: 4vw;
  background-color: #FADADD;
  color: black;

}
```