Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Desarrollo de un sistema Control Supervisor y Adquisición de Datos (SCADA) para la gestión de centros de producción robotizados

Proyecto Fin de Grado

Grado en Ingeniería de Computadores

Curso académico 2024-2025

Autor:
Alberto Iglesias Bravo

Tutor:
Borja Bordel Sánchez

Resumen

En las últimas décadas se ha visto un avance exponencial en la robótica , hasta tal punto que se pueden desarrollar proyectos y funcionalidades complejas con pocos recursos.

El objetivo de este trabajo de fin de grado ha sido el de desarrollar un sistema de control supervisor y adquisición de datos para la gestión remota de una mano robótica mediante un ESP32. Esto se ha llevado a cabo con un servidor local que envía las acciones de una web al microcontrolador, además de poder consultar el estado actual del dispositivo, combinando así de forma eficaz hardware y software.

Abstract

In recent decades, there has been an exponential advancement in robotics, to the point where complex projects and functionalities can be developed with minimal resources.

The objective of this final degree project has been to develop a supervisory control and data acquisition system for the remote management of a robotic hand using an ESP32. This has been achieved through a local server that sends web-based commands to the microcontroller, as well as allows monitoring of the device's current status, effectively combining hardware and software.

Índice

Índice de figuras

Capítulo 1

Introducción

1.1. Introducción

La robótica es una disciplina que ha evolucionado rápidamente en poco tiempo, ayudando al desarrollo de numerosos sectores como la medicina, la industria o la exploración espacial. Gracias a la irrupción de la Inteligencia Artificial y a los avances en sensores y microcontroladores, los robots han pasado de hacer tareas repetitivas previamente programadas a sistemas capaces de adaptarse al cambio y conseguir funcionalidades más complejas.

En la medicina, por ejemplo, es cada vez más útil, como en operaciones con el sistema Da Vinci [1], el cual es un conjunto de brazos robóticos que permite hacer cirugías más precisas, más rápidas y menos invasivas, o en prótesis robóticas que ayudan a las personas con miembros amputados o maltrechos a tener una vida de más calidad.

También se ha implementado muy bien en la industria [2] ya que ayuda a aumentar la eficacia en los procesos, se reduce el tiempo de producción y se aprovechan mejor los materiales, además de que se ofrece información en tiempo real y, con ello, se pueden detectar fallos e incluso solucionarlos desde un lugar diferente al que se encuentra la maquinaria.

Por otro lado, en la exploración espacial, la robótica ha ayudado a investigar otros planetas y entornos fuera de nuestro planeta sin poner en riesgo vidas humanas. Esto ha sido gracias a vehículos como los rovers marcianos [3], que han sido muy importantes para la recolección de datos científicos en territorios desconocidos donde las condiciones serían inviables para los astronautas.

1.2. Objetivos

El objetivo del proyecto es diseñar y desarrollar un sistema que sea capaz de controlar una mano robótica mediante un servidor web. Se busca que el sistema sea eficiente, de bajo costo y que se pueda usar en diferentes ámbitos, como por ejemplo, el educativo o en el desarrollo sanitario de prótesis asequibles.

Para este caso se definen los siguientes objetivos:

- Implementar un sistema de control basado en el ESP32
- Desarrollar una página web en la que se puedan realizar las acciones requeridas y ver el estado actual del dispositivo.
- Implementar correctamente la conexión cliente-servidor.
- Aplicar los conocimientos adquiridos en el grado para la integración de hardware y software en un sistema funcional.

1.3. Estructura del documento

En este documento se reflejará el proceso seguido para la realización del proyecto, así como el funcionamiento de este tanto a nivel de hardware como de software.

Hardware: se verán los dispositivos usados, al igual que la forma de conectarlos para el correcto funcionamiento del sistema.

Software: se divide en tres partes: el código usado por el ESP32, el código que hace que el servidor y la mano se comuniquen y el código de la página web.

GitHub: donde se encuentra el código final y necesario para que funcione el sistema.

Por último, se enseñarán los resultados, problemas durante el proceso y soluciones, además de posibles mejoras.

Capítulo 2

Estado del arte

El origen de la robótica se remonta a la antigua Grecia, donde Aristóteles ya hablaba de 'herramientas automatizadas', aunque los avances más significativos no los encontramos hasta mediados del siglo XX, cuando surgieron robots que sentaron las bases de la robótica actual. Entre ellos destacan:

- **ElMER y ELSIE** [4]: Creados por William Grey Walter en la década de 1940, estos robots estaban formados principalmente por dos motores, uno de avance y uno de giro, y dos sensores, uno de luz y otro de contacto. A estos dispositivos se les considera las primeras máquinas robóticas autónomas de la historia, con ellos su creador quería demostrar que se podían crear máquinas autónomas que pudieran replicar ciertos comportamientos de los seres vivos. De hecho, con un número muy limitado de componentes sensoriales, consiguió que expresaran 4 tipos de comportamientos [4]:
  - Exploración
  - Fototropismo positivo
  - Fototropismo positivo
  - Esquivar obstáculos

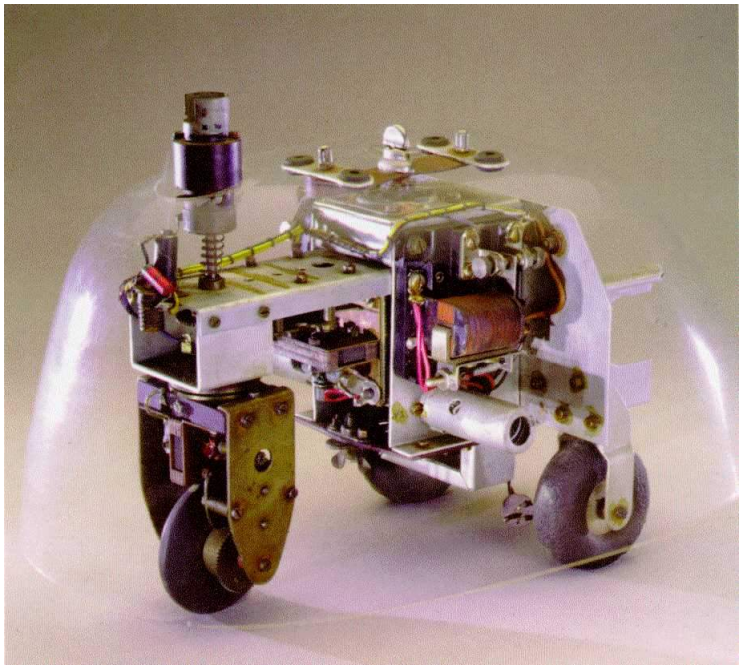


Figura 2.1: ELISE.

- **SHAKEY** [**5**]: Desarrollado en los años 60 por el Stanford Research Institute, este robot era capaz de percibir su entorno, planificar una ruta y desplazarse de forma autónoma. Esto era gracias a la combinación de sensores y cámaras, algo muy avanzado para la época. Se dice que fue "el primer robot inteligente del mundo" y ha sido importante para el diseño de servidores, automóviles o videojuegos.

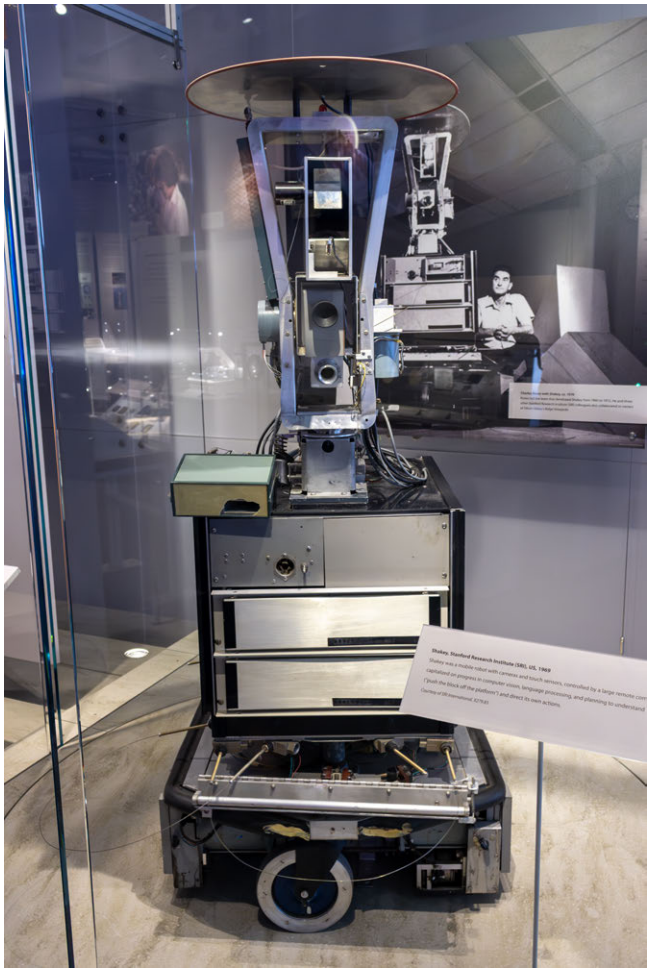


Figura 2.2: SHANKEY.

- **Sojouner[4]**: Fue el primer mars rover de todos, tenía el tamaño de un microondas y pesaba 11,5 kilos, en principio se pensó que podía durar 7 días pudiendo ampliar hasta 30, pero aún así rompió todas las expectativas y cuando perdieron su conexión el día 83, aún seguía activo. Este robot llevaba 2 cámaras monocolor y 1 a color para poder estudiar la composición de las rocas, unos sensores en las ruedas para determinar cómo de abrasivo era el terreno y otro para comprobar la eficacia de los paneles solares que podía verse afectada por la acumulación de polvo.

Figura 2.3: 1er Mars Rover(Sojourner).

Con el avance de la tecnología, se han desarrollado robots que intentan acercarse a las formas y funciones humanas:

- **ASIMO** [**6**]: Creado por Honda en el año 2000, es un robot humanoide capaz de interactuar con personas, caminar, correr y reconocer gestos y voces humanas. Fue el primer robot bípedo capaz de mantener el equilibrio y de subir y bajar escaleras algo que para principios del siglo XXI fue una revolución tecnológica enorme.

  La ultima versión de este robot se presento en 2014, llegaba a los 1,3 metros de altura y 50 kg de peso y llegando a correr 9km/h, ademas de ser capaz de caminar por pendientes irregulares.

Figura 2.4: ASIMO.

- **NAO [7]**: Creado por la empresa francesa Aldebaran Robotics en 2004, se trata de un robot muy beneficioso en varios ámbitos como el educativo, donde fomenta la participación del alumnado, el trabajo en equipo y es útil en los proyectos relacionados con las ciencias y las matemáticas, o en la investigación ya que tiene una plataforma de pruebas para modelos teóricos, pero sobretodo ayuda al desarrollo de niños autistas ya que favorece la interacción de manera visual, verbal y táctil.

  Cuenta con un software en el que se puede utilizar Windows, Linux o MAC. También cuenta con herramientas de programación para controlar su comportamiento, soportando desde una programación a partir de arrastrar bloques mas orientado a niños hasta lenguajes como C++, Java o Python para gente mas experimentada en programación.

Figura 2.5: NAO.

- **TEMI [8]**: Presentado en 2018, es un robot asistente personal que combina inteligencia artificial y movilidad avanzada. En Fuenlabrada, se ha implementado como robot de asistencia domiciliaria para personas mayores, con funciones como mantener conversaciones básicas, realizar llamadas a contactos autorizados, recordar la medicación, monitorizar el estado de salud y localizar al usuario en caso de emergencia.

Figura 2.6: TEMI.

La robótica también ha avanzado en el ámbito de la seguridad y la asistencia en situaciones de riesgo:

- **Robot TEDAX-NRBQ** [**9**]: Incorporado por la Policía Nacional de España, este robot está diseñado para la detección, manipulación y desactivación de explosivos, su uso garantiza un aumento en el nivel de seguridad ya que ayudaran a las Fuerzas y Cuerpos de Seguridad del Estado a prevenir y luchar contra el terrorismo y los delincuentes graves y organizados.

  Estos robots pueden neutralizar tanto amenazas terroristas convencionales, como las nuevas, ya sean bombas sucias, agentes químicos o biológicos y radiaciones ionizantes, todo ello sin exponer directamente a los agentes y minimizando así su riesgo.

Figura 2.7: TEDAX.

La inteligencia artificial ha permitido el desarrollo de robots más autónomos y eficientes:

- **Gemini Robotics [10]**: Google está aprovechando su modelo de inteligencia artificial, Gemini, para su uso tanto en máquinas industriales como en robots humanoides, con el objetivo de lograr comportamientos más humanos y realizar operaciones más complejas.

  Con esta idea han desarrollado dos vías: Gemini Robotics que se enfoca en la visión, el lenguaje y la acción, lo que ayudará a mejorar su respuesta en situaciones dinámicas, y Gemini Robotics-ER, que esta mas enfocado a los expertos en robótica, lo que les da herramientas para poder desarrollar y probar sus propios programas.

  Además, ha sido diseñado para ser ejecutado en distintos tipos de robots. Principalmente se entrenó con ALOHA 2, una plataforma de dos brazos, aunque también ha podido controlar sistemas como los brazos Franka, habituales en laboratorios, e incluso humanoides más avanzados como Apolo.

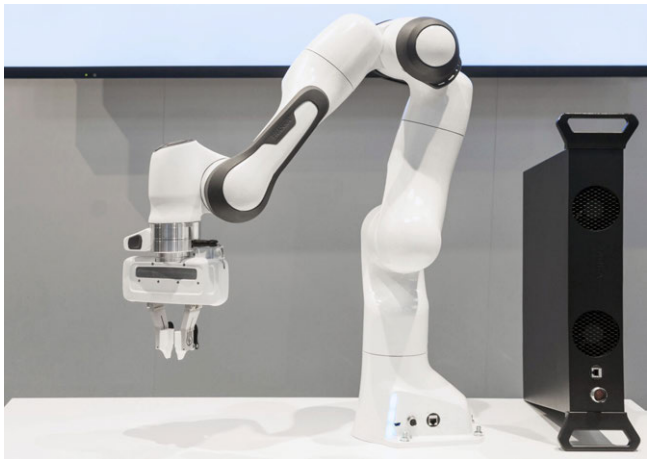


Figura 2.8: Franka Robot.

Figura 2.9: Apolo Robot.

En cuanto a similitudes con el proyecto que he realizado, existen muchas opciones variadas que ayudan a la industria o a personas con miembros amputados:

- **Mano robótica Mia-Hand [11]**: Esta mano ha sido creada por investigadores del Instituto de BioRobótica de la Escuela Superior de Santa Ana (Pisa, Italia).

  El dispositivo funciona con imanes implantados en el antebrazo y mediante el movimiento que genera la contracción del mismo se abren y cierran los dedos, ademas gracias a su interfaz miocinética permitió a su paciente no solo controlar la prótesis sin problemas si no también sentirla como si fuera su propia mano.

Figura 2.10: Mia-Hand.

- **Mano impresa en 3D [12]**: Esta mano robotica ha sido desarrollada por científicos japoneses de la Universidad de Hiroshima y el Instituto Hyogo de Tecnologías de Apoyo.

  Esta mano está fabricada con una impresora 3D, cuanta con 5 dedos móviles de forma independiente y que funciona gracias a la detección de las pequeñas señales eléctricas que se generan en nuestro cuerpo al mover algún musculo.

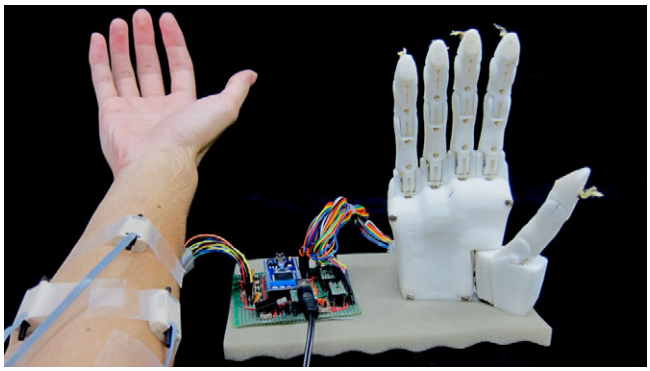


Figura 2.11: Mano impresa en 3D.

- **Automatización logística [13]**: Hoy en día, la industria ha implantado brazos mecánicos autónomos para realizar trabajos repetitivos o peligrosos, reduciendo costos y riesgos laborales.

  Estos brazos pueden ser de varios tipos:

  - **Brazo robótico articulado**: Este brazo imita la estructura de un brazo humano gracias a múltiples segmentos unidos por juntas giratorias. Suele utilizarse en ensamblaje, soldadura, pintura, entre otras.
  - **Brazo robótico cartesiano**: Estos brazos siguen las coordenadas cartesianas para su rango de movimiento. Son muy útiles para impresión 3D y manipulación precisa.
  - **Brazo robótico cilíndrico**: Se mueven principalmente de forma vertical, estos brazos se usan principalmente como máquinas de control numérico y manipulación de materiales a granel.
  - **Brazo robótico SCARA**: Los brazos robóticos SCARA (Selective Compliance Assembly Robot Arm) están diseñados para ensamblajes rápidos y precisos y son perfectos para montajes.
  - **Brazo robótico delta**: Son robots rapidos y precisos en paralelo, son ideales para la seleccion y colocacion de objetos en las lineas de producción.

Figura 2.12: Ejemplo de uso de brazos roboticos en logistica.

Capítulo 3

Arquitectura

El objetivo del proyecto era desarrollar una aplicación web desde la que se pudiera controlar de forma remota un centro de producción robotizado.

3.1. Requisitos

Los requisitos mínimos para cumplir el objetivo son:

- La mano debe ser capaz de mover todos los dedos tanto de forma individual como en colectivo.
- El ESP32 debe comunicarse correctamente con el servidor.
- Debe haber una interfaz web que incluya todas las funcionalidades y el estado actual del dispositivo.
- El dispositivo contará con rutinas diferentes controladas por el usuario.

3.2. Arquitectura

Hardware:

- **ESP32**: Microcontrolador que cuenta con Wi-Fi y pines para el correcto funcionamiento del sistema.
- **Servomotores**: 5 servomotores encargados de mover cada dedo.
- **Mano robótica 5 grados de libertad QDS-1601**: Estructura necesaria para simular la forma de la mano.

Software:

- **ESP32**: Software desarrollado en el IDE de Arduino.
- **Aplicación web**: Aplicación donde se encuentran diferentes botones e información requerida.

3.3. Desarrollo

En cuanto al desarrollo, primero se realizó el montaje de la mano robótica y la conexión de los servomotores para comprobar su correcto funcionamiento, donde se probaron tanto de forma individual como colectiva.

El siguiente paso fue empezar con el desarrollo del software, donde se puso en marcha un programa local con rutinas simples, añadiendo después una página HTML y conexión a través del ESP32 a ella, para acabar con un servidor local haciendo uso de NodeJS.

Tras todo lo anterior, se realizaron pruebas para comprobar que la comunicación entre el servidor y el ESP32 era correcta, además de añadir un apartado donde se mostraba el estado actual de la mano y algunas rutinas más complejas. Durante este proceso se encontraron problemas relacionados con el ángulo necesario para cerrar los dedos, lo que fue ajustado en la programación.

Capítulo 4

Interfaz de Usuario

Esta aplicación está desarrollada en JavaScript y HTML, ya que es una página web. Para utilizarla, será necesario introducir en el buscador la IP del dispositivo que ejecuta el servidor.

Al abrir la página nos encontraremos con lo siguiente:

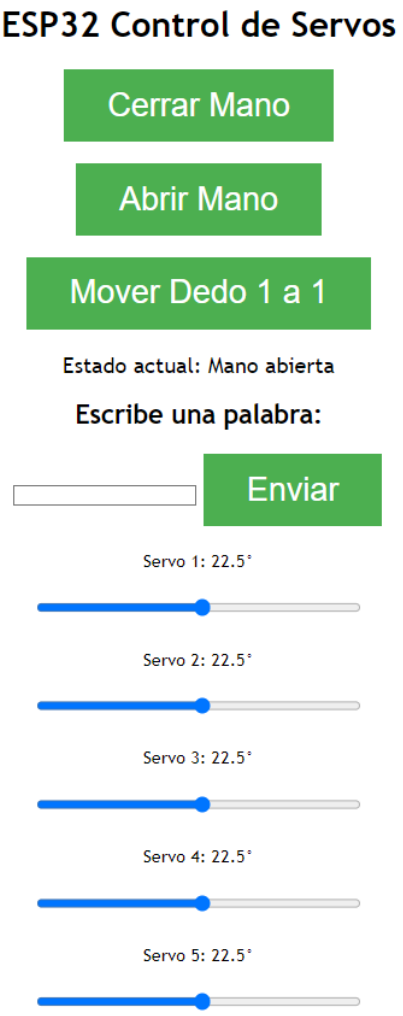


Figura 4.1: Vista de la web.

En ella se pueden distinguir varias secciones:

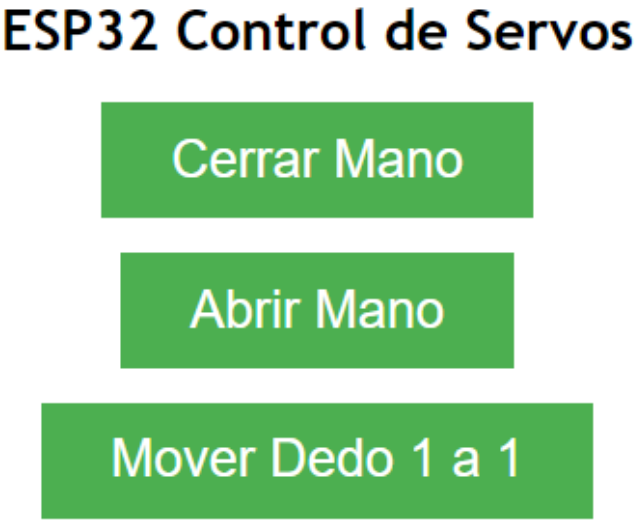


Figura 4.2: Rutinas sencillas.

Aquí se encuentra el título y tres rutinas sencillas que son:

- **Cerrar Mano**: Cierra los 5 dedos.
- **Abrir Mano**: Abre los 5 dedos.
- **Mover Dedo 1 a 1**: Va cerrando y abriendo los dedos de forma secuencial.

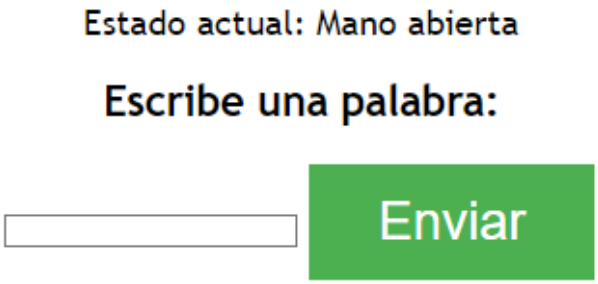


Figura 4.3: Cuadro de texto y estado.

Estado y rutina compleja:

- **Estado Actual**: Refleja el estado en el que se encuentra la mano (por defecto se encuentra abierta).
- **Escribe una Palabra**: Esta parte lo que hace es que al enviar una palabra, ésta se manda al ESP32 para que el microcontrolador le indique a la mano que la deletree en lenguaje de signos.

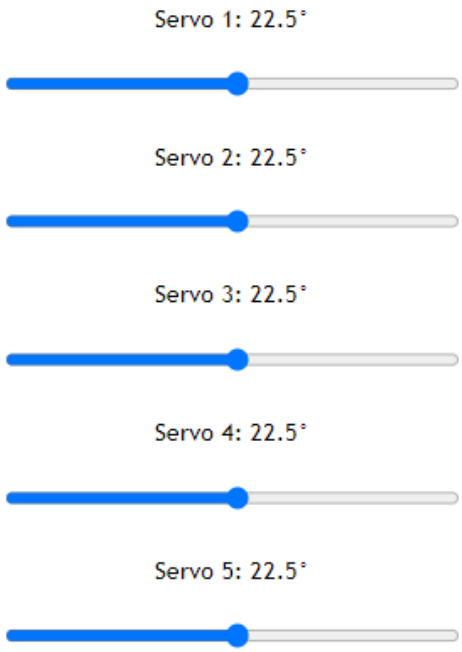


Figura 4.4: Sliders.

Sliders: sirven para mover cada dedo de forma individual con un rango de 0º a 45º.

Capítulo 5

Hardware

5.1. Descripción de los componentes

El sistema implementado se basa en una mano robótica prefabricada, que se montó siguiendo las instrucciones proporcionadas. Los componentes que conforman el hardware son:

- **Estructura de la mano**: Conjunto de piezas metálicas que deja un espacio en el centro para poder insertar los servomotores.

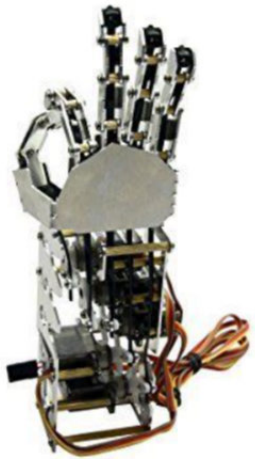


Figura 5.1: Mano Física.

- **Servomotores**: 5 servomotores, 1 por cada dedo que, con ayuda de una brida, controla el angulo de los dedos.

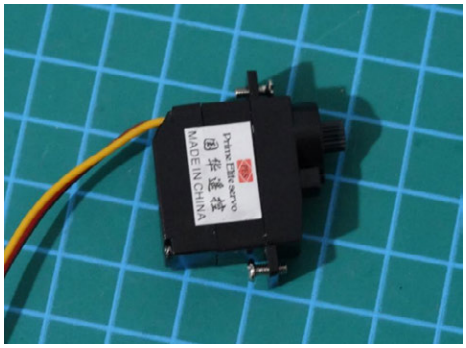


Figura 5.2: Servomotores.

- **ESP32**: Microcontrolador que se encarga de recibir las acciones del servidor y trasladarla a los servomotores.

Figura 5.3: ESP32.

- **Cables**: 3 cables usados por cada dedo necesarios para recibir corriente, para la conexión a tierra y para poder transmitir la señal desde el ESP32 al servomotor.

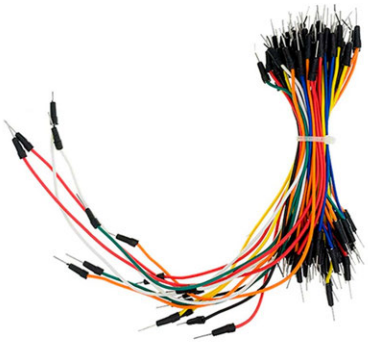


Figura 5.4: Cables.

5.2. Implementación hardware

5.2.1. Montaje

Para el montaje de la estructura de la mano se siguieron las instrucciones del fabricante, cuyo proceso es:

1. Se arman los dedos.

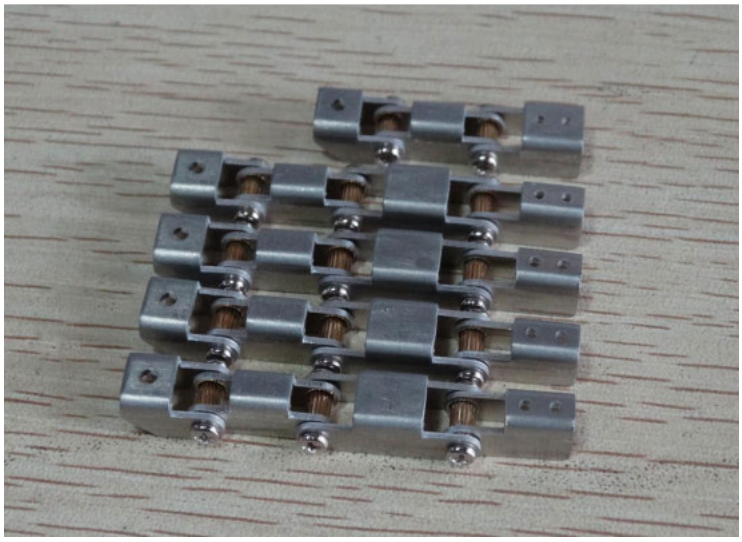


Figura 5.5: Montaje dedos.

2. Se monta la estructura donde van a ir colocados los servomotores.

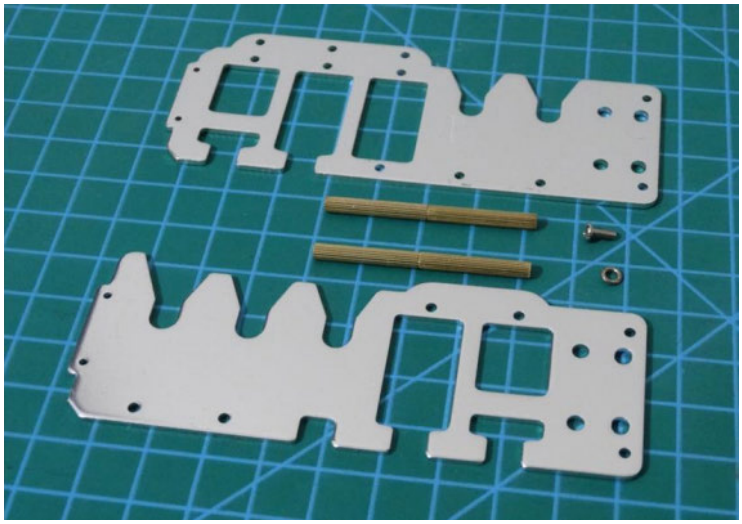


Figura 5.6: Piezas usadas.

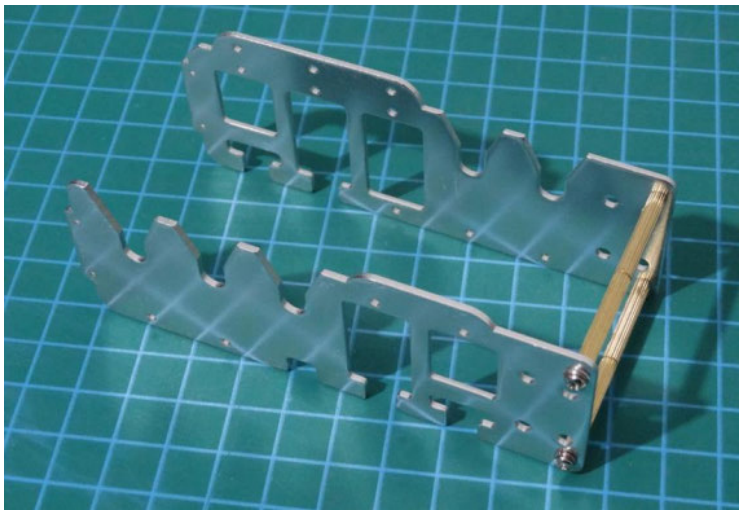


Figura 5.7: Estructura Montada.

3. Se unen los dedos con la mano.

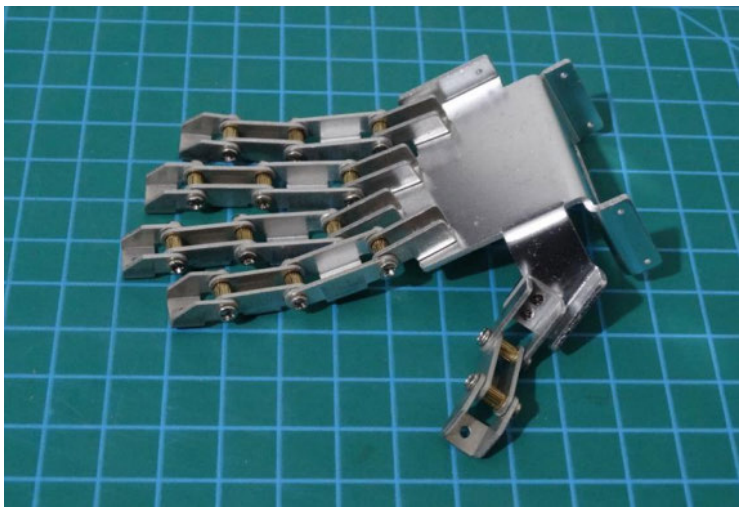


Figura 5.8: Dedos unidos.

4. Se une la estructura con la mano.

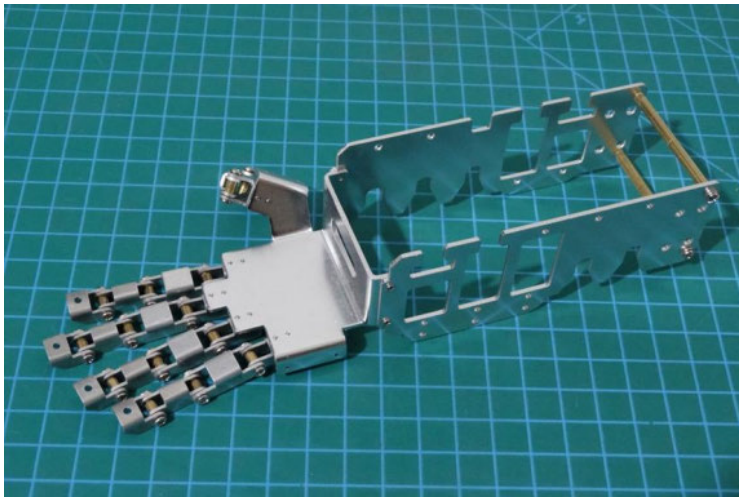


Figura 5.9: Estructura Completa.

5. Se montan las piezas necesarias para unir los servos con la estructura.

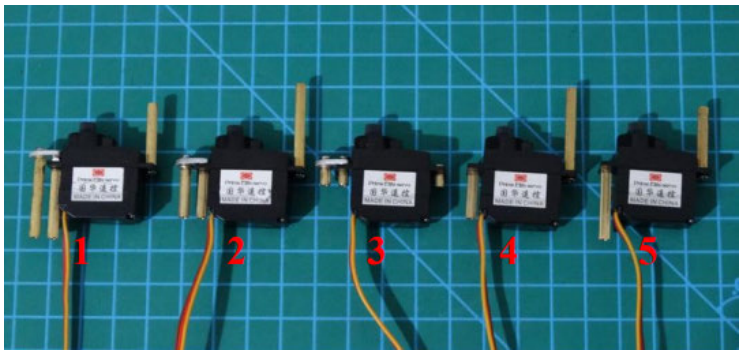


Figura 5.10: Servos con las piezas.

6. Se montan los servos en la estructura.

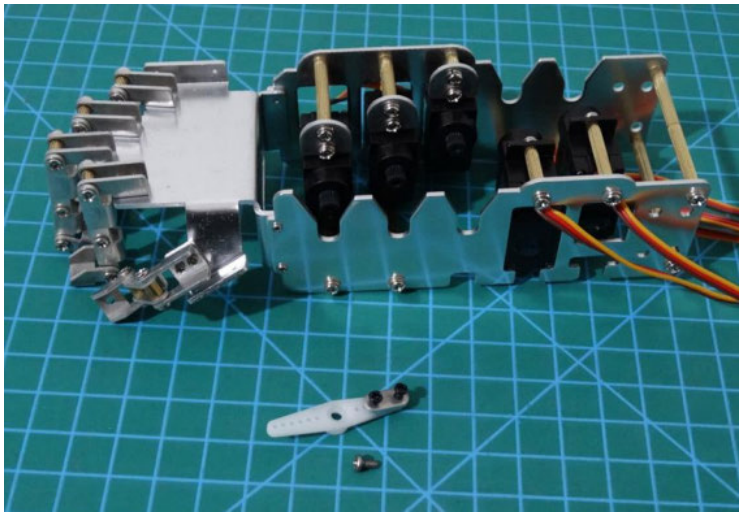


Figura 5.11: Servos montados.

7. Se colocan las piezas que unirán los servos con los dedos.

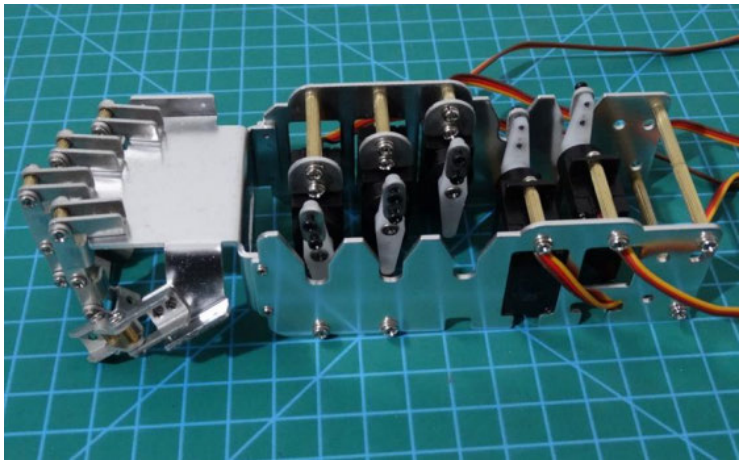


Figura 5.12: Servos montados en la estructura.

8. Por ultimo se ponen y cortas unas bridas en los dedos y se unen a los servos.

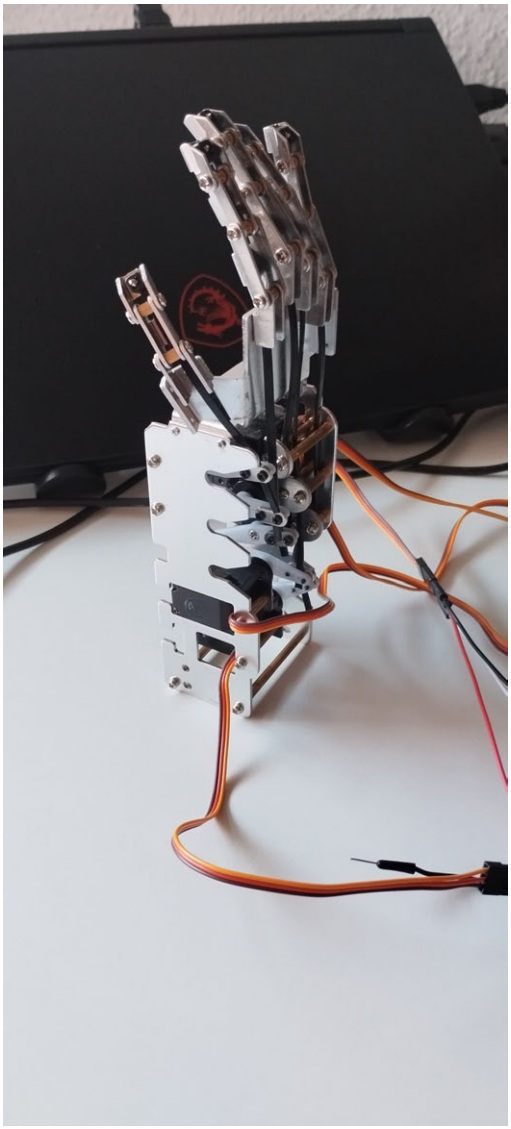


Figura 5.13: Mano final montada.

5.2.2. Funcionamiento del sistema

Los servos tienen 3 cables: el de alimentación, el de tierra y el de señal.

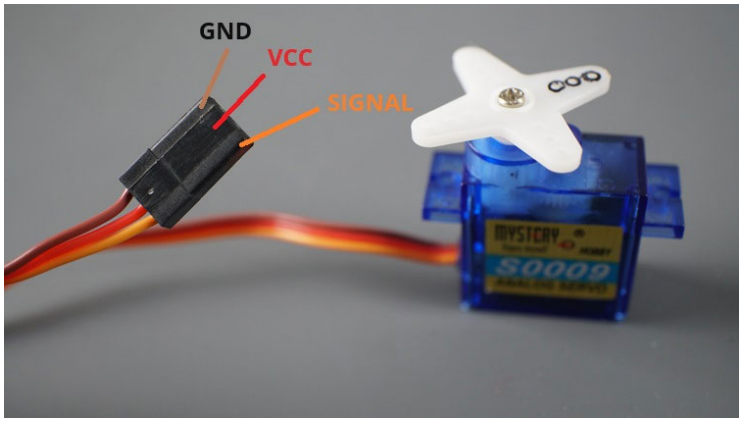


Figura 5.14: Conexiones Servo.

Para el correcto funcionamiento del sistema se conectará cada servo de la siguiente forma:

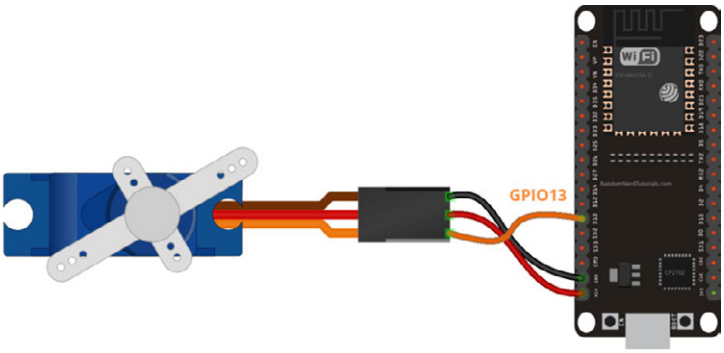


Figura 5.15: Conexión ESP32 con servomotor.

El cable de señal de cada servo debe ir conectado a un pin que sea capaz de transmitir una señal PWM, señal que hará que cambien los grados de rotación del servo.

Capítulo 6

Software

6.1. Diseño Software

El proyecto está diseñado con 3 tipos de software: por un lado el que controla el ESP32; por otro el que controla el servidor; y por último, el de la interfaz web. Cada parte de software tenía que cumplir con los siguientes requisitos:

6.1.1. Requisitos

ES32
En cuanto al software del ESP32, se necesitaba:

- Que pudiera controlar los servomotores, tanto de forma individual como colectiva
- Que fuera capaz de recibir acciones de un servidor.
- Que imprimiese por consola las acciones realizadas para control de errores

Servidor
El servidor sería el encargado de recibir las instrucciones del usuario a través de la interfaz web para después enviárselas al ESP32. Para ello, los requisitos serían:

- Que hubiera una correcta comunicación entre el servidor y el ESP32
- Que las acciones realizadas desde la interfaz web fueran capaces de enviarse al controlador

Web
Para realizar las acciones era necesaria una interfaz web donde se elige qué quiere hacer la mano, para esto, los requisitos serían:

- Permitir al usuario controlar a su gusto cada dedo de forma individual.
- Mostrar el estado en el que se encontraba en ese momento el dispositivo.
- Ofrecer algunas funcionalidades predefinidas que se ejecutarán con solo pulsar un botón (abrir mano, deletrear palabras,etc.)

6.1.2. Estructuras y diagramas

En este apartado se reflejan los diagramas de flujo del servidor y el ESP32.

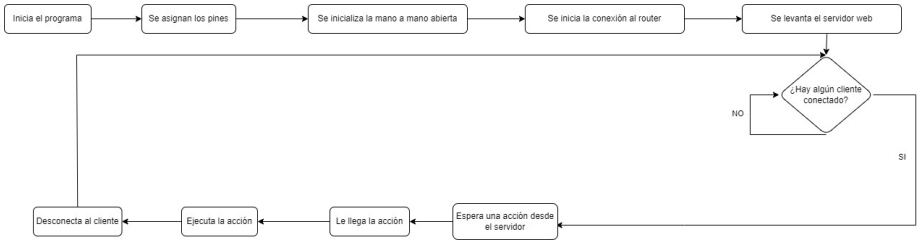


Figura 6.1: Diagrama de flujo del software de la ESP32.

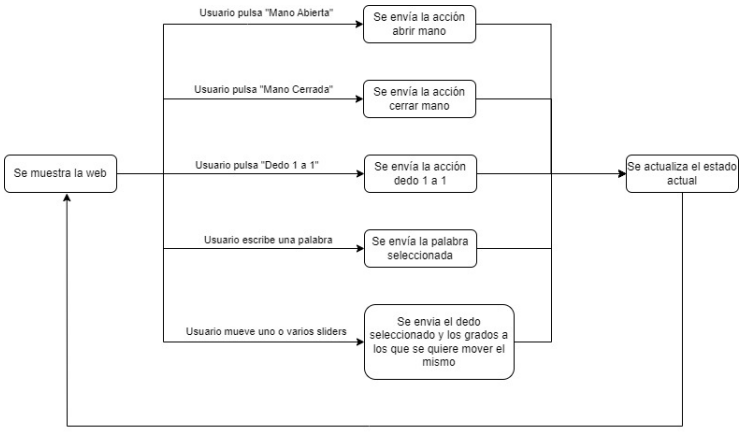


Figura 6.2: Diagrama de flujo de la aplicación web.

6.2. Implementación Software

En esta sección se explicará el código utilizado para el funcionamiento del sistema, tanto del ESP32 como de la página web o el JavaScript del servidor.

6.2.1. ESP32

El ESP32 ha sido programado en el Arduino IDE, ya que el lenguaje Arduino es simple pero efectivo para el control de servomotores a través de un servidor web.

```
#include <WiFi.h>
#include <ESP32Servo.h>


Servo miMenique;
Servo miAnular;
Servo miMedio;
Servo miIndice;
Servo miPulgar;


static const int anular26 = 26;
static const int menique27 = 27;
static const int medio25 = 25;
static const int indice33 = 33;
static const int pulgar32 = 32;
```

Figura 6.3: Declaraciones.

En la imagen se observan las librerías importadas y la asignación de pines para los dedos:

- **WiFi.h**: Con esta librería se consiguen las funciones necesarias para que el ESP32 se conecte a internet.
- **ESP32Servo.h**: Esta librería es necesaria para el control de servomotores, que son los responsables del movimiento de los dedos.

A continuación, se puede ver la asignación de pines de cada dedo:

- Se declaran 5 objetos servos con su nombre correspondiente a cada dedo existente.
- El nombre de cada dedo está asignado a un número, que corresponde al pin físico donde debe conectarse para que funcione correctamente.

```
const char* ssid = "SSID Router";
const char* password = "Contraseña";


WiFiServer server(80);
String header;
```

Figura 6.4: Declaraciones wifi.

En la imagen superior se encuentran las declaraciones y funciones necesarias para conectarse a Internet:

- Primero se declaran dos variables `ssid` y `password` que contienen el nombre de la red a utilizar y su contraseña.
- Se declara un objeto `WiFiServer` llamado `server`, que escuchará en el puerto 80 (el puerto estándar para HTTP). Esto permite que el microcontrolador actúe como un servidor web.
- Se declara una variable `header` de tipo `String` que almacenará el contenido de la solicitud HTTP que el servidor reciba.

```
void setup() {
  Serial.begin(115200);


  miMenique.attach(menique27);
  miAnular.attach(anular26);
  miMedio.attach(medio25);
  miIndice.attach(indice33);
  miPulgar.attach(pulgar32);



  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());
  server.begin();
}
```

Figura 6.5: Setup.

Aquí se encuentra la función `setup()` que se encarga de la configuración inicial del sistema:

- Se inicia la comunicación serial a `115200` baudios para depuración y monitoreo desde el puerto serie.
- Se asignan los pines GPIO a cada uno de los servomotores que controlan los dedos.
- Se inicia la conexión WiFi con las credenciales definidas (`ssid` y `password`).
- Se muestra en el Monitor Serie que la conexión está en progreso.
- Se usa un `while` para esperar hasta que la conexión sea exitosa (`WL_CONNECTED`).
- Una vez conectado, se imprime la dirección IP asignada al ESP32 en la red.
- Se inicia el servidor web en el puerto `80` para escuchar peticiones HTTP.

```
void mano_cerrada() {
  miMenique.write(0);
  miAnular.write(0);
  miMedio.write(20);
  miIndice.write(20);
  miPulgar.write(20);
  delay(3000);
}

void mano_abierta() {
  miMenique.write(20);
  miAnular.write(20);
  miMedio.write(0);
  miIndice.write(0);
  miPulgar.write(0);
  delay(3000);
}

void dedo_1a1() {
  miMenique.write(20);
  delay(1000);
  miAnular.write(20);
  delay(1000);
  miMedio.write(0);
  delay(1000);
  miIndice.write(0);
  delay(1000);
  miPulgar.write(0);
  delay(2000);

  miMenique.write(0);
  delay(1000);
  miAnular.write(0);
  delay(1000);
  miMedio.write(20);
  delay(1000);
  miIndice.write(20);
  delay(1000);
  miPulgar.write(20);
  delay(3000);
}
```

Figura 6.6: Rutinas sencillas.

Después de lo mencionado anteriormente, se encuentran las funciones para las rutinas predefinidas:

- **Mano_cerrada**: Cierra la mano completamente.
- **Mano_abierta**: Abre la mano completamente.
- **Dedo_1a1**: Los dedos se abren y cierran de forma secuencial.

En todas ellas se observa que dos dedos tienen un valor diferente a los otros tres. Esto se debe a que, por la disposición física de los servomotores, esos dos están colocados en el sentido contrario.

```
void moverServo(Servo& servo, int posicion) {
  servo.write(posicion);
  delay(15);
}
```

Figura 6.7: Movimiento individual.

Aquí aparece la función que se usa para mover los dedos de forma individual mediante los sliders, lo que hará que se lea qué servo va a mover y la posición deseada.

```
void mostrar_letra(String letra){
  letra.toLowerCase();
  switch(letra[0]){
    case 'a':
      miMenique.write(0);
      miAnular.write(0);
      miMedio.write(45);
      miIndice.write(45);
      miPulgar.write(45);
      delay(3000);
      Serial.println("Letra A");
      break;
    case 'b':
      miMenique.write(20);
      miAnular.write(20);
      miMedio.write(0);
      miIndice.write(0);
      miPulgar.write(45);
      delay(3000);
      Serial.println("Letra B");
      break;
    case 'c':
      miMenique.write(5);
      miAnular.write(5);
      miMedio.write(15);
      miIndice.write(15);
      miPulgar.write(15);
      delay(3000);
      Serial.println("Letra C");
      break;
    case 'd':
      miMenique.write(20);
      miAnular.write(5);
      miMedio.write(20);
      miIndice.write(20);
      miPulgar.write(20);
      delay(3000);
      Serial.println("Letra D");
      break;
```

Figura 6.8: Función de lenguaje de signos.

Esta función es la que almacena todas las letras del abecedario en lenguaje de signos y la que cuando le llega una letra comprueba qué dedos mover y de qué forma mediante un switch.

```
void loop() {
  WiFiClient client = server.available();

  if (client) {
    Serial.println("Nuevo cliente conectado.");
    String currentLine = "";
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        header += c;

        if (c == '\n') {
          if (currentLine.length() == 0) {

            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();


            if (header.indexOf("GET /mano_cerrada") >= 0) {
              mano_cerrada();
              Serial.println("Ejecutando: Mano Cerrada");
            } else if (header.indexOf("GET /mano_abierta") >= 0) {
              mano_abierta();
              Serial.println("Ejecutando: Mano Abierta");
            } else if (header.indexOf("GET /dedo_1a1") >= 0) {
              dedo_1a1();
              Serial.println("Ejecutando: Mover Dedo 1 a 1");
            } else if (header.indexOf("GET /servo") >= 0) {

              int servoNum = header.substring(header.indexOf("/servo") + 6, header.indexOf("?")).toInt();
              int pos = header.substring(header.indexOf("pos=") + 4).toInt();
```

Figura 6.9: Función loop parte 1.

Este es el `loop()`, que es el encargado de manejar la conexión con clientes web y de ejecutar acciones en función de las solicitudes recibidas:

- **Espera conexiones entrantes**:
  - `server.available()` verifica si hay un cliente conectado al servidor web. Si hay un cliente, se crea un objeto `WiFiClient` llamado `client` para manejar la comunicación.
- **Si hay un cliente conectado**:
  - Se imprime un mensaje en el Monitor Serie.
  - Se inicializa `currentLine`, que se usará para leer la solicitud HTTP.
  - Se entra en un `while` que mantiene la conexión mientras el cliente siga conectado.
- **Si el cliente ha enviado datos**:
  - Se lee carácter por carácter (`client.read()`) y se almacena en `header`, que contiene la solicitud HTTP completa.
  - Cada línea de la solicitud HTTP termina en `n`.
  - Cuando se encuentra una línea vacía (`currentLine.length() == 0`), esto significa que terminó la solicitud HTTP del cliente.

- **Análisis de la solicitud HTTP**:
  - Si la solicitud contiene `"GET /mano_cerrada"`, se ejecuta `mano_cerrada()`, que mueve los servos para cerrar la mano.
  - Si la solicitud contiene `"GET /mano_abierta"`, se ejecuta `mano_abierta()`.
  - Si la solicitud contiene `"GET /dedo_1a1"`, se ejecuta `dedo_1a1()`, que mueve cada dedo de manera individual.
  - Si la solicitud contiene `"GET /servo"`, se extraen dos valores:

```
            switch (servoNum) {
              case 1:
                moverServo(miPulgar, pos);
                client.println("Servo 1 movido a posición " + String(pos));
                break;
              case 2:
                moverServo(miIndice, pos);
                client.println("Servo 2 movido a posición " + String(pos));
                break;
              case 3:
                moverServo(miMedio, pos);
                client.println("Servo 3 movido a posición " + String(pos));
                break;
              case 4:
                moverServo(miAnular, pos);
                client.println("Servo 4 movido a posición " + String(pos));
                break;
              case 5:
                moverServo(miMenique, pos);
                client.println("Servo 5 movido a posición " + String(pos));
                break;
              default:
                client.println("Número de servo no válido");
                break;
            }
            Serial.printf("Servo %d movido a posición %d\n", servoNum, pos);
          } else if (header.indexOf("GET /palabra?texto=") >= 0) {
            int start = header.indexOf("texto=") + 6;
            String palabra = header.substring(start, header.indexOf(" ", start));
            for (int i = 0; i < palabra.length(); i++) {
              mostrar_letra(String(palabra[i]));
              delay(1000);
            }
            Serial.println("Ejecutando: Deletrear palabra " + palabra);
          }

          break;
        } else {
          currentLine = "";
        }
      } else if (c != '\r') {
        currentLine += c;
      }
    }
  }
  header = "";
  client.stop();
  Serial.println("Cliente desconectado.");
 }
}
```

Figura 6.10: Función loop parte 2.

- **servoNum**: El número del servo (dedo) que se va a mover.
- **pos**: La posición a la que se moverá.
- Según el número del servo, se llama a `moverServo()` con la posición indicada.
- Se envía una respuesta al cliente con la confirmación.
- Si la solicitud es `"GET /palabra?texto=..."`, se extrae la palabra enviada. Se recorre la palabra letra por letra, ejecutando `mostrar_letra()`, que mueve los servos para representar la letra en lenguaje de señas.

- Una vez procesada la solicitud, se termina el bucle y se limpia `header`.
- Se cierra la conexión con el cliente (`client.stop()`) y se imprime en el Monitor Serie un mensaje que anuncia que el cliente se ha desconectado.

6.2.2. Servidor

Para el servidor se ha usado Node.js que usa JavaScript, el código es el siguiente:

```
const express = require('express');
const axios = require('axios');
const path = require('path');
const app = express();
const port = 8000;

const ESP32_IP = 'http://192.168.1.29';

let estadoActual = 'Mano abierta';

app.use(express.static(path.join(__dirname)));

app.get('/estado', (req, res) => {
  res.send(estadoActual);
});

app.get('/mano_cerrada', async (req, res) => {
  try {
    await axios.get(`${ESP32_IP}/mano_cerrada`);
    estadoActual = 'Mano cerrada';
    console.log('Comando enviado: Mano cerrada');
    res.send('Mano cerrada');
  } catch (error) {
    console.error('Error enviando comando:', error);
    res.status(500).send('Error enviando comando al ESP32');
  }
});

app.get('/mano_abierta', async (req, res) => {
  try {
    await axios.get(`${ESP32_IP}/mano_abierta`);
    estadoActual = 'Mano abierta';
    console.log('Comando enviado: Mano abierta');
    res.send('Mano abierta');
  } catch (error) {
    console.error('Error enviando comando:', error);
    res.status(500).send('Error enviando comando al ESP32');
  }
});
```

Figura 6.11: Servidor Parte 1.

- **express**: Crea el servidor web.
- **axios**: Permite enviar solicitudes HTTP al ESP32.
- **path**: Facilita el manejo de rutas de archivos.
- Se inicializa Express.
- Se establece el puerto 8000 para escuchar solicitudes.

- Se define la dirección IP local del ESP32 (debe estar en la misma red que el servidor).
- Se usa una variable para almacenar el estado actual de la mano robótica.
- Permite servir archivos estáticos desde la carpeta donde está el script.
- Por ejemplo, si hay un archivo `index.html` en el mismo directorio, se podrá acceder desde el navegador.
- Cuando un cliente, por ejemplo, un navegador, accede a `/estado`, el servidor responde con el estado actual de la mano. Cuando un cliente accede a `/mano_cerrada`:
  - Se envía un `GET` al ESP32 en `http://192.168.1.24/mano_cerrada`.
  - Si la solicitud es exitosa:
    - Se actualiza `estadoActual` a "Mano cerrada".
    - Se imprime en la consola.
    - Se responde "Mano cerrada" al cliente.
  - Si hay un error, se imprime en la consola y se responde con un error 500.
- Similar a la ruta anterior, pero envía un `GET` a `http://192.168.1.24/mano_abierta` para abrir la mano.
- El usuario accede a una de las rutas en el navegador (ejemplo: `http://localhost:8000/mano_cerrada`).
- El servidor Node.js recibe la solicitud y usa Axios para enviar un comando HTTP al ESP32.
- El ESP32 recibe el comando y mueve los servos.
- Node.js actualiza el estado de la mano y responde al cliente.

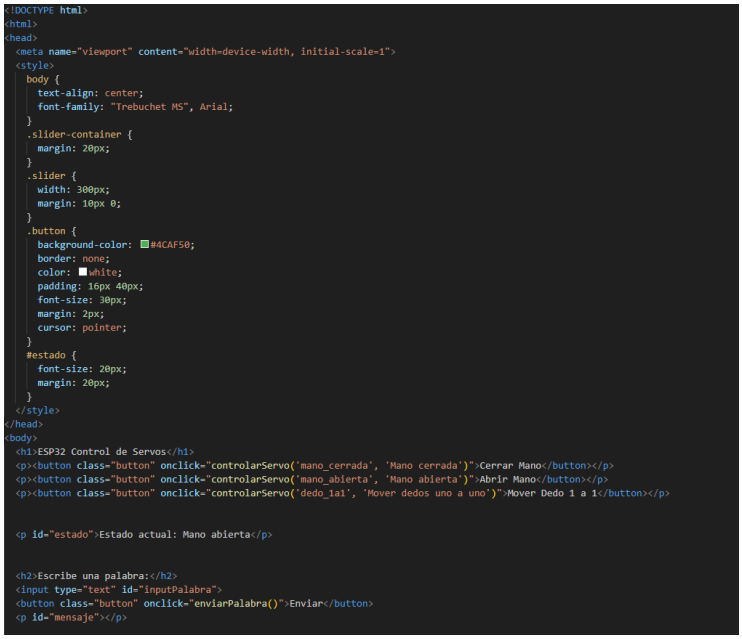
```
app.get('/dedo_1a1', async (req, res) => {
  try {
    await axios.get(`${ESP32_IP}/dedo_1a1`);
    estadoActual = 'Mover dedos uno a uno';
    console.log('Comando enviado: Mover dedos uno a uno');
    res.send('Mover dedos uno a uno');
  } catch (error) {
    console.error('Error enviando comando:', error);
    res.status(500).send('Error enviando comando al ESP32');
  }
});

app.get('/servo:servo', async (req, res) => {
  const { servo } = req.params;
  const { pos } = req.query;
  try {
    await axios.get(`${ESP32_IP}/servo${servo}?pos=${pos}`);
    console.log(`Comando enviado: Mover servo ${servo} a posición ${pos}`);
  } catch (error) {
    console.error('Error enviando comando:', error);
    res.status(500).send('Error enviando comando al ESP32');
  }
});


app.get('/palabra', async (req, res) => {
  const { texto } = req.query;
  estadoActual = 'Deletrear palabra';
  try {
    await axios.get(`${ESP32_IP}/palabra?texto=${texto}`);
    console.log(`Comando enviado: Deletrear palabra ${texto}`);
    res.send(`Palabra ${texto} enviada`);
  } catch (error) {
    console.error('Error enviando comando:', error);
    res.status(500).send('Error enviando comando al ESP32');
  }
});

app.listen(port, () => {
  console.log(`Servidor Express escuchando en http://localhost:${port}`);
});
```

Figura 6.12: Servidor Parte 2.

- **Acceso**: `http://localhost:8000/dedo_1a1`
- **Función**:
  - Envía un comando `GET` al ESP32 en la ruta `/dedo_1a1`.
  - Actualiza `estadoActual` a "Mover dedos uno a uno".
  - Imprime el mensaje en la consola.
  - Responde al cliente con "Mover dedos uno a uno".
- **Acceso**: `http://localhost:8000/servo1?pos=90`
- **Función**:
  - Usa `req.params` y `req.query` para capturar el número del servo y la posición deseada.
  - Envía un `GET` al ESP32 con la URL.
  - Actualiza `estadoActual` a "Deletrear palabra".
  - Imprime en la consola el comando enviado.
  - Responde "Palabra hola enviada" al cliente.
- Inicia el servidor en el puerto 8000.
- Imprime en la consola que el servidor está listo.

6.2.3. Pagina web

Para la página web se usa HTML y el código es el siguiente:



```html
<!DOCTYPE html>
<html>
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <style>
    body {
      text-align: center;
      font-family: "Trebuchet MS", Arial;
    }
    .slider-container {
      margin: 20px;
    }
    .slider {
      width: 300px;
      margin: 10px 0;
    }
    .button {
      background-color: #4CAF50;
      border: none;
      color: white;
      padding: 16px 40px;
      font-size: 30px;
      margin: 2px;
      cursor: pointer;
    }
    #estado {
      font-size: 20px;
      margin: 20px;
    }
  </style>
</head>
<body>
  <h1>ESP32 Control de Servos</h1>
  <p><button class="button" onclick="controlarServo('mano_cerrada', 'Mano cerrada')">Cerrar Mano</button></p>
  <p><button class="button" onclick="controlarServo('mano_abierta', 'Mano abierta')">Abrir Mano</button></p>
  <p><button class="button" onclick="controlarServo('dedo_1a1', 'Mover dedos uno a uno')">Mover Dedo 1 a 1</button></p>


  <p id="estado">Estado actual: Mano abierta</p>


  <h2>Escribe una palabra:</h2>
  <input type="text" id="inputPalabra">
  <button class="button" onclick="enviarPalabra()">Enviar</button>
  <p id="mensaje"></p>
```

Figura 6.13: Estilo, botones y cuadro de texto web.

El documento HTML sigue el estándar HTML5 y define una página web responsive para su visualización en distintos dispositivos. La sección ¡head¿incluye una metaetiqueta para ajustar la escala en dispositivos móviles y una hoja de estilos CSS para mejorar la apariencia de los elementos de la interfaz.

Se emplean estilos CSS para mejorar la experiencia del usuario:

- **Fuente y alineación**: Se establece una tipografía clara y un diseño centrado.
- **Botones de control**: Se diseñan con un fondo verde, texto en blanco y un tamaño adecuado para facilitar la interacción.
- **Indicador de estado**: Se muestra el estado actual de la mano robótica con un tamaño de fuente legible.
- **Sliders**: Se configuran controles deslizantes para ajustar la posición de los servos de manera precisa.

La interfaz permite diferentes tipos de interacción:

- **Botones de control**: Permiten cerrar y abrir la mano, así como mover los dedos individualmente.
- **Indicador de estado**: Muestra el estado actual de la mano, actualizándose dinámicamente según las acciones del usuario.

- **Campo de entrada de texto**: Permite escribir una palabra para que la mano intente representarla.

```
<div class="slider-container">
  <p>Servo 1: <span id="posServo1">22.5</span>°</p>
  <input type="range" min="0" max="45" value="22.5" class="slider" id="sliderServo1" oninput="moverServo(1, this.value)">
</div>

<div class="slider-container">
  <p>Servo 2: <span id="posServo2">22.5</span>°</p>
  <input type="range" min="0" max="45" value="22.5" class="slider" id="sliderServo2" oninput="moverServo(2, this.value)">
</div>

<div class="slider-container">
  <p>Servo 3: <span id="posServo3">22.5</span>°</p>
  <input type="range" min="0" max="45" value="22.5" class="slider" id="sliderServo3" oninput="moverServo(3, this.value)">
</div>

<div class="slider-container">
  <p>Servo 4: <span id="posServo4">22.5</span>°</p>
  <input type="range" min="0" max="45" value="22.5" class="slider" id="sliderServo4" oninput="moverServo(4, this.value)">
</div>

<div class="slider-container">
  <p>Servo 5: <span id="posServo5">22.5</span>°</p>
  <input type="range" min="0" max="45" value="22.5" class="slider" id="sliderServo5" oninput="moverServo(5, this.value)">
</div>

<script>
  let estadoActual = "Mano abierta";

  window.onload = function() {
    fetch('/estado')
      .then(response => response.text())
      .then(data => {
        estadoActual = data;
        document.getElementById("estado").textContent = `Estado actual: ${estadoActual}`;
      })
      .catch(error => {
        console.error('Error:', error);
        document.getElementById("estado").textContent = "Error al cargar el estado actual.";
      });
  };
```

Figura 6.14: Sliders y estado actual.

Esta parte del código define cinco deslizadores (input type=range") que permiten controlar individualmente los cinco servomotores de la mano robótica. Cada deslizador tiene un valor inicial de 22.5° y un rango de movimiento de 0° a 45°. Cuando el usuario mueve un deslizador, se ejecuta la función moverServo(), que envía la nueva posición al servidor para actualizar la posición del servo correspondiente.

Además, se incluye un fragmento de código JavaScript que se ejecuta al cargar la página (window.onload). Esta función realiza una petición al servidor (fetch('/estado')) para obtener el estado actual de la mano y actualizar la interfaz con la última configuración guardada. Si la solicitud es correcta, se actualiza el texto del estado; en caso de error, se muestra un mensaje indicando que no se pudo obtener la información.

Este mecanismo asegura que, al abrir la interfaz, el usuario pueda ver la posición real de los servos y ajustar su movimiento de forma precisa.

```
let debounceTimers ={};
function moverServo(servo, valor) {
  estadoActual = "Personalizado";
  document.getElementById("estado").textContent = `Estado actual: ${estadoActual}`;
  if (servo === 4 || servo === 5) {
    valor = 45 - valor;
  }

  document.getElementById(`posServo${servo}`).textContent = valor;

  clearTimeout(debounceTimers[servo]);
  debounceTimers[servo] = setTimeout(() => {
    fetch(`/servo${servo}?pos=${valor}`)
      .then(response => response.text())
      .then(data => {
        document.getElementById("estado").textContent = `Servo ${servo}: ${data}`;
      })
      .catch(error => {
        console.error('Error:', error);
        document.getElementById("estado").textContent = "Error al comunicarse con el servidor.";
      });
  }, 500);
}


function controlarServo(comando, nuevoEstado) {
  estadoActual = nuevoEstado;
  document.getElementById("estado").textContent = `Estado actual: ${estadoActual}`;
  fetch(`/${comando}`)
    .then(response => response.text())
    .then(data => {
      console.log(`Comando ejecutado: ${comando}`);
    })
    .catch(error => {
      console.error('Error:', error);
      document.getElementById("estado").textContent = "Error al comunicarse con el servidor.";
    });
}
```

Figura 6.15: Funciones moverServo y controlarServo.

Función moverServo(servo, valor)

Esta función se ejecuta cuando el usuario ajusta un deslizador para modificar la posición de un servomotor. Su objetivo es actualizar la interfaz con el nuevo valor y enviar la instrucción al ESP32 para mover el servo.

Cambia el estado actual:

- Se actualiza la variable `estadoActual` con el valor "Personalizado"para indicar que el usuario ha realizado un ajuste manual.
- Se actualiza el texto en la página para reflejar este cambio.

Ajuste de valores para ciertos servos:

- Si el servo es el 4 o el 5, se invierte el valor (`valor = 45 - valor`).
- Esto se debe a que estos servos están montados en una orientación diferente y requieren una inversión de los valores de ángulo.

Actualiza la interfaz:

- Se modifica el contenido del elemento `<span>` correspondiente para reflejar el nuevo valor del servo.

Envía la instrucción al ESP32:

- Se usa `fetch()` para hacer una solicitud HTTP al servidor con la nueva posición del servo (`/servoX?pos=valor`).
- Si la petición es exitosa, se muestra el mensaje `"Servo X: respuesta del servidor"`.

- Si hay un problema en la comunicación, se muestra un mensaje de error en la interfaz.

Control de peticiones:

- Para evitar saturaciones del servidor por peticiones al cambiar los ángulos mediante el slider se ha implementado un sistema de debounce.
- Cada vez que se molifica el valor existe un temporizador de 500 ms que espera ese tiempo hasta enviar la petición, con ello se asegura que se envía la posición que realmente quiere el usuario.

Función controlarServo(comando, nuevoEstado)
Esta función se activa cuando el usuario pulsa uno de los botones predefinidos, como Abrir Mano o Cerrar Mano.

Actualiza el estado en la interfaz:

- La variable `estadoActual` toma el valor de `nuevoEstado`, que es el texto asociado a la acción ejecutada.
- Se actualiza el texto en la página para reflejar el nuevo estado.

Envía el comando al ESP32:

- Se realiza una solicitud HTTP al servidor con el comando correspondiente (`fetch('/comando')`).
- Si el servidor responde correctamente, se muestra `"Comando ejecutado: comando"`.
- En caso de error, se muestra un mensaje de advertencia.

Estas funciones garantizan una interacción fluida entre el usuario y la mano robótica, permitiendo controlar los servos tanto mediante botones como a través de ajustes manuales con los deslizadores.

```
function enviarPalabra() {
  var palabra = document.getElementById("inputPalabra").value.toLowerCase();
  if (palabra.trim() === "") {
    document.getElementById("mensaje").textContent = "Por favor, introduce una palabra.";
    return;
  }

  estadoActual = "Deletrear palabra";
  document.getElementById("estado").textContent = `Estado actual: ${estadoActual}`;

  fetch(`/palabra?texto=${palabra}`)
    .then(response => response.text())
    .then(data => {
      document.getElementById("mensaje").textContent = `Enviando: ${palabra}`;
    })
    .catch(error => {
      console.error("Error al enviar palabra:", error);
      document.getElementById("mensaje").textContent = "Error al enviar la palabra.";
    });
}
</script>
</body>
</html>
```

Figura 6.16: Función enviarPalabra.

Obtención y validación de la palabra:

- Se obtiene el valor ingresado por el usuario en el campo de texto con `document.getElementById(ïnputPalabra").value`.
- La palabra se convierte a minúsculas utilizando `.toLowerCase()` para asegurarse de que no haya diferencias entre mayúsculas y minúsculas.
- Si el campo de entrada está vacío (comprobado con `trim()` para eliminar estos posibles espacios), se muestra un mensaje pidiendo al usuario que introduzca una palabra.

Actualización del estado:

- Si la palabra es válida, se cambia el estado actual a "Deletrear palabra", indicando que el sistema está listo para realizar el proceso de deletreo con la mano robótica.
- Se actualiza la interfaz para reflejar este nuevo estado.

Envío de la palabra al servidor:

- Se utiliza `fetch()` para enviar la palabra al servidor a través de una solicitud HTTP (`/palabra?texto=palabra`).
- Si la solicitud es exitosa, se muestra un mensaje indicando que la palabra se está enviando y se presenta en la interfaz.
- Si hay algún fallo en la comunicación con el servidor, se muestra un mensaje de error.

Capítulo 7

Aspectos medioambientales, sociales y económicos

- **Eficiencia energética**: El ESP32 es un microcontrolador que consume poca electricidad, lo que ayuda a reducir el gasto energético en comparación con otros sistemas más complejos.
- **Reducción de desplazamientos**: Al tener la posibilidad del control a distancia y la visualización del estado de la mano, solo habría que ir al lugar físico donde se encuentra el dispositivo en caso de urgencia, lo que reduce la huella de carbono asociada al transporte.
- **Accesibilidad**: La posibilidad de controlar una mano robótica a distancia puede ser muy útil en ámbitos como la asistencia a personas con discapacidad o en tareas donde exista un riesgo para la persona que las realiza.
- **Aplicaciones en educación y formación**: Puede servir como una herramienta de enseñanza para estudiantes que quieran aprender sobre robótica, control de sistemas embebidos y automatización.
- **Seguridad en entornos industriales**: Permite operar dispositivos en lugares peligrosos sin poner en riesgo a los trabajadores.
- **Reducción de costes**: El uso de un hardware asequible como el ESP32 y tecnologías de código abierto permiten desarrollar una solución económica en comparación con sistemas industriales comerciales.
- **Útil en pequeñas empresas y startups**: Al utilizar materiales y componentes de bajo coste, puede ser útil en empresas con recursos limitados que necesiten soluciones de automatización sin grandes inversiones.
- **Mantenimiento y mejoras**: Al ser un sistema basado en componentes sencillos y tecnologías accesibles, su mantenimiento y mejoras futuras se pueden realizar sin grandes inversiones.

Capítulo 8

Github y pruebas

Todo el código desarrollado durante el proyecto, así como vídeos de cada una de las funcionalidades, se encuentran en GitHub, en `https://github.com/Albertoib01/Alberto_Iglesias_Bravo_TFG`

Figura 8.1: Repositorio del código en GitHub.

El repositorio contiene seis carpetas, que reflejan las distintas fases del desarrollo del proyecto:

- **1ª Prueba**: Código básico para probar diferentes rutinas simples. Contiene las declaraciones de los dedos y las funciones "Abrir Mano", "Cerrar Mano", "Dedo 1 a 1". Fue usada para entender el funcionamiento de los servos.

```
#include <ESP32Servo.h>

Servo miMenique;
Servo miAnular;
Servo miMedio;
Servo miIndice;
Servo miPulgar;

#define menique 27
#define anular 26
#define medio 25
#define indice 33
#define pulgar 32


String output27State = "off";
String output26State = "off";
String output25State = "off";
String output33State = "off";
String output32State = "off";

const int output26 = 26;
const int output27 = 27;
const int output25 = 25;
const int output33 = 33;
const int output32 = 32;
```

Figura 8.2: Declaraciones.

```
void mano_cerrada(){
  miMenique.write(15);
  miAnular.write(15);
  miMedio.write(15);
  miIndice.write(0);
  miPulgar.write(0);

  delay(3000);

}

void mano_abierta(){
  miMenique.write(0);
  miAnular.write(0);
  miMedio.write(0);
  miIndice.write(15);
  miPulgar.write(15);

  delay(3000);

}

void dedo_1a1(){
  miMenique.write(20);
  delay(1000);
  miAnular.write(20);
  delay(1000);
  miMedio.write(0);
  delay(1000);
  miIndice.write(0);
  delay(1000);
  miPulgar.write(0);
  delay(2000);

  miMenique.write(0);
  delay(1000);
  miAnular.write(0);
  delay(1000);
  miMedio.write(20);
  delay(1000);
  miIndice.write(20);
  delay(1000);
  miPulgar.write(20);

  delay(3000);
}


void loop(){
  dedo_1a1();
}
```

Figura 8.3: Rutinas.

- **2º Paso Wi-Fi**: Conexión al ESP32 a través de Wi-Fi y envío de acciones desde una web. Se añade una conexión al router y en el loop un código HTML y la correspondiente gestión de peticiones con una pagina a mostrar al conectarse al microcontrolador.

```
WiFiServer server(80);
String header;

void setup() {
  Serial.begin(115200);


  miMenique.attach(menique27);
  miAnular.attach(anular26);
  miMedio.attach(medio25);
  miIndice.attach(indice33);
  miPulgar.attach(pulgar32);


  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
    Serial.println(WiFi.status());
  }

  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  server.begin();
}
```

Figura 8.4: Declaraciones.

```
void loop() {
  WiFiClient client = server.available();

  if (client) {
    Serial.println("New Client.");
    String currentLine = "";
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
        header += c;

        if (c == '\n') {
          if (currentLine.length() == 0) {

            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();


            client.println("<!DOCTYPE html><html>");
            client.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">");
            client.println("<style>body { text-align: center; font-family: \"Trebuchet MS\", Arial; }");
            client.println(".button { background-color: #4CAF50; border: none; color: white; padding: 16px 40px; font-size: 30px; margin: 2px; cursor: pointer;}");
            client.println(".button2 {background-color: #555555;}</style></head>");

            client.println("<body><h1>ESP32 Control de Servos</h1>");


            client.println("<p><a href=\"/mano_cerrada\"><button class=\"button\">Cerrar Mano</button></a></p>");


            client.println("<p><a href=\"/mano_abierta\"><button class=\"button\">Abrir Mano</button></a></p>");

            client.println("<p><a href=\"/dedo_1a1\"><button class=\"button\">Mover Dedo 1 a 1</button></a></p>");


            client.println("</body></html>");


            if (header.indexOf("GET /mano_cerrada") >= 0) {
              mano_cerrada();
              Serial.println("Ejecutando: Mano Cerrada");
            }
            else if (header.indexOf("GET /mano_abierta") >= 0) {
              mano_abierta();
              Serial.println("Ejecutando: Mano Abierta");
            }
            else if (header.indexOf("GET /dedo_1a1") >= 0) {
              dedo_1a1();
              Serial.println("Ejecutando: Mover Dedo 1 a 1");
            }

            client.println();
            break;
          } else {
            currentLine = "";
          }
        } else if (c != '\r') {
          currentLine += c;
        }
      }
    }
    header = "";
    client.stop();
    Serial.println("Client disconnected.");
  }
}
```

Figura 8.5: Loop.

- **3º Paso Node.js**: Levantamiento de un servidor local con Node.js para enviar acciones al ESP32. Cambio del loop para poder gestionar las peticiones desde un servidor local que estará fuera del ESP32.

```
void loop() {
  WiFiClient client = server.available();

  if (client) {
    Serial.println("Nuevo cliente conectado.");
    String currentLine = "";
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        header += c;

        if (c == '\n') {
          if (currentLine.length() == 0) {

            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();



            if (header.indexOf("GET /mano_cerrada") >= 0) {
              mano_cerrada();
              Serial.println("Ejecutando: Mano Cerrada");
            } else if (header.indexOf("GET /mano_abierta") >= 0) {
              mano_abierta();
              Serial.println("Ejecutando: Mano Abierta");
            } else if (header.indexOf("GET /dedo_1a1") >= 0) {
              dedo_1a1();
              Serial.println("Ejecutando: Mover Dedo 1 a 1");
            }

            break;
          } else {
            currentLine = "";
          }
        } else if (c != '\r') {
          currentLine += c;
        }
      }
    }
    header = "";
    client.stop();
    Serial.println("Cliente desconectado.");
  }
}
```

Figura 8.6: Loop.

- **4º Paso Mejoras**: Inclusión de sliders para mover los dedos de forma individual.

  Se añade una nueva función que se capaz de mover un servo elegido a un angulo especificado, añadiendo un switch en el loop que ayude a completar esta funcionalidad.

```
void moverServo(Servo& servo, int posicion) {
  servo.write(posicion);
  delay(15);
}
```

Figura 8.7: Funcion moverServo.

```
    switch (servoNum) {
      case 1:
        moverServo(miPulgar, pos);
        client.println("Servo 1 movido a posición " + String(pos));
        break;
      case 2:
        moverServo(miIndice, pos);
        client.println("Servo 2 movido a posición " + String(pos));
        break;
      case 3:
        moverServo(miMedio, pos);
        client.println("Servo 3 movido a posición " + String(pos));
        break;
      case 4:
        moverServo(miAnular, pos);
        client.println("Servo 4 movido a posición " + String(pos));
        break;
      case 5:
        moverServo(miMenique, pos);
        client.println("Servo 5 movido a posición " + String(pos));
        break;
      default:
        client.println("Número de servo no válido");
        break;
    }
    Serial.printf("Servo %d movido a posición %d\n", servoNum, pos);
}
```

Figura 8.8: Switch moverServo.

- **ESP32**: Código final con las rutinas anteriores y capacidad de deletrear palabras en lenguaje de signos.

  Se añade una nueva función que gestione el movimiento de los dedos segun la letra elegida y se cambia el loop para que pueda gestionar esta nueva petición.

```
void mostrar_letra(String letra){
  letra.toLowerCase();
  switch(letra[0]){
    case 'a':
      miMenique.write(0);
      miAnular.write(0);
      miMedio.write(45);
      miIndice.write(45);
      miPulgar.write(45);
      delay(3000);
      Serial.println("Letra A");
      break;
    case 'b':
      miMenique.write(20);
      miAnular.write(20);
      miMedio.write(0);
      miIndice.write(0);
      miPulgar.write(45);
      delay(3000);
      Serial.println("Letra B");
      break;
    case 'c':
      miMenique.write(5);
      miAnular.write(5);
      miMedio.write(15);
      miIndice.write(15);
      miPulgar.write(15);
      delay(3000);
      Serial.println("Letra C");
      break;
    case 'd':
      miMenique.write(20);
      miAnular.write(5);
      miMedio.write(20);
      miIndice.write(20);
      miPulgar.write(20);
      delay(3000);
      Serial.println("Letra D");
      break;
```

Figura 8.9: Función mostrarLetra.

```
} else if (header.indexOf("GET /palabra?texto=") >= 0) {
  int start = header.indexOf("texto=") + 6;
  String palabra = header.substring(start, header.indexOf(" ", start));
  for (int i = 0; i < palabra.length(); i++) {
    mostrar_letra(String(palabra[i])); // Mostrar cada letra en la mano
    delay(1000); // Pausa entre letras
  }
  Serial.println("Ejecutando: Deletrear palabra " + palabra);
}
```

Figura 8.10: Gestión mostrarLetra.

- **Server web**: Código JavaScript y HTML del servidor y la interfaz web, respectivamente.

Para acabar, hay una lista de reproducción donde hay videos en los que se demuestra el funcionamiento del sistema: `https://www.youtube.com/watch?v=Kd5IVLRjsGU&list=PLjbuJDVMNRE6Khx6iuLZO1hgfFNQFE-xN&index=1`

Capítulo 9

Resultados

9.1. Resultados obtenidos

Al probar el sistema, se ha comprobado su funcionamiento de la siguiente forma:

- **El servidor envía correctamente las órdenes al ESP32**: Cada vez que se realiza una acción en la página web, el terminal del servidor muestra correctamente la acción enviada.
- **El ESP32 recibe correctamente las acciones del servidor**: El terminal de Arduino muestra la acción a realizar, coincidiendo con la acción enviada desde el servidor.
- **La mano es capaz de realizar rutinas simples**: La mano se abre y se cierra a voluntad del usuario, además de poder mover los dedos uno a uno, ya sea de forma secuencial o individual.
- **La mano es capaz de deletrear una palabra en lenguaje de signos**: La mano recibe de la página web una palabra a deletrear en lenguaje de signos, siguiendo el abecedario del Ministerio de Educación y Formación Profesional.[14].
- **Estado actual del dispositivo**: La web es capaz de mostrar qué función está realizando la mano y cambia correctamente cuando se selecciona una nueva función.

9.2. Objetivos logrados

- **Implementar un sistema de control basado en el ESP32, elegido por su conectividad y capacidad de procesamiento**: Se ha desarrollado un sistema en el que el ESP32 recibe instrucciones desde el servidor y ejecuta dichas instrucciones de forma eficaz.
- **Desarrollo de una página web en la que se puedan realizar las acciones requeridas y ver el estado actual del dispositivo**: Se ha creado una interfaz web, accesible desde cualquier navegador, que permite al usuario realizar diferentes acciones con la mano y conocer el estado de la misma en cada momento.
- **Implementar correctamente la conexión cliente-servidor**: Se ha establecido una comunicación estable entre el ESP32 y el servidor, consiguiendo así que la acción a realizar por la mano se ejecute de forma eficaz.

- **Aplicar los conocimientos adquiridos en el grado para la integración de hardware y software en un sistema funcional**: Se han aplicado los conocimientos de electrónica, programación y redes para diseñar un sistema que integra hardware y software, logrando un sistema funcional y adaptable.

9.3. Problemas encontrados

A lo largo del proyecto, surgieron diferentes obstáculos, tanto con el hardware como con el software. Las principales dificultades fueron:

- **Dificultad a la hora del montaje**: Entre las piezas incluidas para el montaje de la mano, algunas eran muy similares, lo que llevó a desmontarla y volver a montarla en varias ocasiones, ya que pequeñas diferencias en la longitud de las piezas podían provocar que algunas partes quedaran forzadas o mal unidas.
- **Comunicación servidor/ESP32**: Se intentaron varias formas de programar el servidor sin éxito hasta llegar a la solución final.
- **Limitación física de la mano**: Hay ciertas limitaciones en los dedos, ya que solo pueden abrirse y cerrarse y no pueden moverse lateralmente. Además, la muñeca es fija, lo que dificulta el deletreo de palabras en lenguaje de signos.
- **Fallo con los sliders**: Al hacer las pruebas finales se detecto que los sliders eran capaces de saturar el servidor ya que lo que hacia era leer todos los valores que por los que pasaba un slider sin importar cual era el valor final deseado.

Capítulo 10

Conclusiones y trabajos futuros

10.1. Conclusiones

Al finalizar el proyecto, se han analizado los resultados obtenidos, comparándolos con productos comerciales y realizando una reflexión sobre los logros personales alcanzados.

El objetivo principal del proyecto era diseñar y desarrollar un sistema capaz de controlar una mano robótica mediante un servidor web. En este sentido, se ha logrado implementar el sistema según lo planeado. Sin embargo, para que sea un producto viable en el mercado, sería necesario disponer de un hardware más flexible y añadir funcionalidades de software más avanzadas, lo que permitiría mejorar sus prestaciones.

En la comparación con productos comerciales, se observa que existen soluciones más avanzadas. No obstante, este proyecto ofrece una alternativa de bajo coste y fácilmente replicable. Aun así, el dispositivo presenta ciertas limitaciones en la movilidad de los dedos y la muñeca, aspectos cuya mejora aumentaría significativamente su funcionalidad y posibilidades de desarrollo.

Desde un punto de vista personal, este proyecto me ha permitido aplicar gran parte del conocimiento adquirido durante el grado. Además, ha sido una oportunidad para enfrentarme a la resolución de problemas técnicos y de integración entre hardware y software, aprendiendo a superar desafíos como la optimización del código, la mejora de la conectividad del sistema y la depuración de errores. También me ha permitido desarrollar habilidades en la gestión del tiempo y en la toma de decisiones para priorizar las funcionalidades más relevantes.

10.2. Líneas futuras

A pesar de lo logrado en el desarrollo del proyecto, hay apartados en los que se puede mejorar para tener más funcionalidades y hacerlo más global. Algunas de las posibles mejoras podrían ser:

- **Añadir dispositivos con más libertad de movimiento**: La mano usada solo puede abrir y cerrar los dedos. En futuras versiones, se podrían añadir dispositivos menos restrictivos para poder desarrollar más funcionalidades.
- **Usar una web a nivel global en vez de local**: Actualmente, el sistema está desarrollado en un servidor local. Con un sistema más complejo, sería ideal tener este servidor en la nube para poder ser controlado desde cualquier parte.
- **Hacer una app para teléfonos móviles**: Aunque la web permite controlar la mano, sería útil desarrollar una aplicación móvil para no depender de un ordenador, además de poder añadir notificaciones en tiempo real.

- **Conseguir que mediante imágenes imite el movimiento de una mano real**: Una gran mejora sería implementar modelos de inteligencia artificial o machine learning para que, mediante imágenes o videos, ya sea en tiempo real o previamente grabados, el sistema pueda imitar los gestos realizados.

Bibliografía

[1] U. Europea, "Uso y aplicación de los robots en la medicina," https://universidadeuropea.com/blog/robots-medicina/, 9/02/2024.

[2] Computing, "Robótica industrial: Qué es, usos y aplicaciones," https://www.computing.es/informes/robotica-industrial-que-es-usos-aplicaciones/, 14/06/2023.

[3] I. Robotics and A. Society, "Space robotics," https://www.ieee-ras.org/space-robotics.

[4] E. B. School, "Robots antiguos e historia de la robótica," https://www.esneca.com/blog/robots-antiguos-historia-robotica/, 25/06/2023.

[5] E. Diario, "Shakey, el primer robot inteligente de la historia y el abuelo del coche autónomo," https://www.eldiario.es/hojaderouter/tecnologia/shakey-robot-inteligencia-artificial-coche-autonomo_1_3466717.html, 11/04/2017.

[6] Honda, "Asimo," https://www.honda.mx/asimo.

[7] Revista de Robots, "Robot humanoide nao; características y precio en 2020," https://revistaderobots.com/robots-y-robotica/robot-nao-caracteristicas-y-precio/, 8/06/2023.

[8] A. de Fuenlabrada, "Temi, primer robot de asistencia domiciliaria," https://www.ayto-fuenlabrada.es/web/portal/w/temi-primer-robot-de-asistencia-domiciliaria-13062024, 13/06/2024.

[9] Ministerio del Interior, "Robots desactivaciÓn tedax y robot gei," https://fondoseuropeosparaseguridad.interior.gob.es/es/detalle/proyecto/ROBOTS-DESACTIVACION-TEDAX-Y-ROBOT-GEI/.

[10] C. Parada, "Gemini robotics brings ai into the physical world," https://deepmind.google/discover/blog/gemini-robotics-brings-ai-into-the-physical-world/, 12/03/2025.

[11] A. SINC, "Desarrollan la primera mano robótica con control magnético," https://www.agenciasinc.es/Noticias/Desarrollan-la-primera-mano-robotica-con-control-magnetico, 12/09/2024.

[12] Público, "Desarrollan una mano robótica muy precisa que requiere un entrenamiento mínimo," https://www.publico.es/ciencias/desarrollan-mano-robotica-precisa-requiere-entrenamiento-minimo.html, 26/06/2019.

[13] T. Forklifts, "Brazo robótico industrial: Qué es, cómo funciona y sus aplicaciones," https://blog.toyota-forklifts.es/brazo-robotico-industrial-para-que-sirve, 11/12/2024.

[14] F. CNSE, "Banco de imágenes y signos. dactilología," https://www.fundacioncnse.org/educa/bancolse/dactilologico.php.