UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Sistema de Control y Supervisión Remoto de un Brazo Robótico

PROYECTO FIN DE GRADO

Guillermo Carrasco Roncero
Grado en Ingeniería de Computadores

Madrid, 2025

UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Grado en Ingeniería de Computadores

Sistema de Control y Supervisión Remoto de un Brazo Robótico

PROYECTO FIN DE GRADO

Guillermo Carrasco Roncero
Grado en Ingeniería de Computadores

Bajo la dirección de:
Dr. Borja Bordel Sánchez

Madrid, 2025

Título: Sistema de Control y Supervisión Remoto de un Brazo Robótico

Autor: Guillermo Carrasco Roncero

Grado en Ingeniería de Computadores

Dirección:

Dr. Borja Bordel Sánchez, ETSISI

Resumen

La robótica está a la orden del día, estando presente tanto en entornos de producción como en la vida cotidiana. Su uso permite facilitar y acelerar procesos manteniendo una gran precisión. Este proyecto está motivado por el desarrollo continuo de la robótica, diferenciándose al implementar un sistema de control remoto mediante interfaces digitales.

El objetivo principal de este trabajo es desarrollar un sistema de control y supervisión de un brazo robótico a través de una aplicación web. Este sistema permite el control individual de cada una de las articulaciones y el desarrollo de secuencias de movimiento que facilitan el traslado y posicionamiento de objetos. Todo ello, permitiendo visualizar el estado del robot en tiempo real a través de la aplicación, mientras que se mantiene una base de datos capacitada para almacenar la información del sistema.

Para cumplir dichos objetivos se ha utilizado un microcontrolador programado para manejar el intercambio de mensajes y actuar sobre el brazo articulado. La aplicación web es alojada en un servidor de NodeJS que permite la constante comunicación y actualización sobre su interfaz. Además, se ha integrado una base de datos con InfluxDB en la que se guarda la información relevante del dispositivo robótico y del servidor. Estas tres partes permanecen conectadas a un broker MQTT el cual facilita el intercambio de mensajes de manera persistente.

De esta forma, se ha conseguido un sistema que facilita el manejo de un brazo robótico con una comunicación instantánea entre la aplicación web y los actuadores, sin generar latencia. Además, este proyecto se alinea con los Objetivos de Desarrollo Sostenible impulsando un crecimiento tecnológico responsable teniendo en cuenta su impacto social y medioambiental.

Abstract

Robotics has become an essential part of both manufacturing and everyday life. Its use facilitates and speed ups processes while maintaining a high level of precision. This project is motivated by the continuous development of robotics, distinguishing itself by implementing a remote-control system through digital interfaces.

The main goal of this project is to develop a system capable of controlling and supervising a robotic arm through a web application. This system allows the individual control of each joint and the execution of movement sequences that facilitate the transport and placement of objects. All of this, allowing to display the robot's status in real time through the web page and the display of a database capable of storing the system's information.

To achieve these goals, a programmed microcontroller has been used to handle the message exchange and control the articulated arm. The web application is hosted in a NodeJS server, which allows a constant communication and the update on its interface. Moreover, a database has also been integrated into the system using InfluxDB to store relevant information from the robotic device and the web server. These three segments remain connected to a MQTT broker which facilitates the exchange of messages in a persistent manner.

This approach has resulted in a system that facilitates the control of a robotic arm with instant communication between the web application and the robotic device. Furthermore, this project aligns with the Sustainable Development Goals by promoting a responsible technological growth considering its social and environmental impact.

Tabla de Contenido

Lista de Figuras

Lista de Tablas

Abreviaturas y Acrónimos

IFR	International Federation of Robotics
MQTT	Message Queuing Telemetry Transport
R.U.R	Rossum's Universal Robots
SCARA	Selective Compliant Assembly Robot Arm
ISS	International Space Station
DOF	Degrees Of Freedom
PWM	Pulse Width Modulation
STA	Station
AP	Access Point
GPIO	General Purpose Input/Output
I2C	Inter-Integrated Circuit
SPI	Serial Peripheral Interface
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
IP	Internet Protocol
HTML	HyperText Markup Language
CPU	Central Processing Unit
API	Application Programming Interface

1. Introducción

1.1. Contexto

El crecimiento de la tecnología ha conllevado amplias mejoras en distintos entornos de trabajo. Mediante la implementación de sistemas robóticos, se ha permitido optimizar el desempeño de diversas tareas gracias a su precisión y velocidad. A través de estos sistemas se han automatizado procedimientos, facilitando y mejorando tareas que antes eran tediosas para los trabajadores.

Las empresas apuestan por la transformación tecnológica como medida para diferenciarse de la competencia, estableciendo sistemas que les permiten una mayor eficiencia con un coste, por lo general, más bajo. Según los datos administrados por la International Federation of Robotics (IFR), en 2024 se superó un nuevo record de robots instalados en fábricas de todo el mundo, superando los cuatro millones de unidades. Estableciendo el foco en España, la industria automotriz supone el principal motor de crecimiento en el desarrollo robótico. Una muestra de ello es la instalación de 5.053 nuevas unidades en cadenas de montaje, lo que representa un incremento del 31% respecto al dato anterior [1].

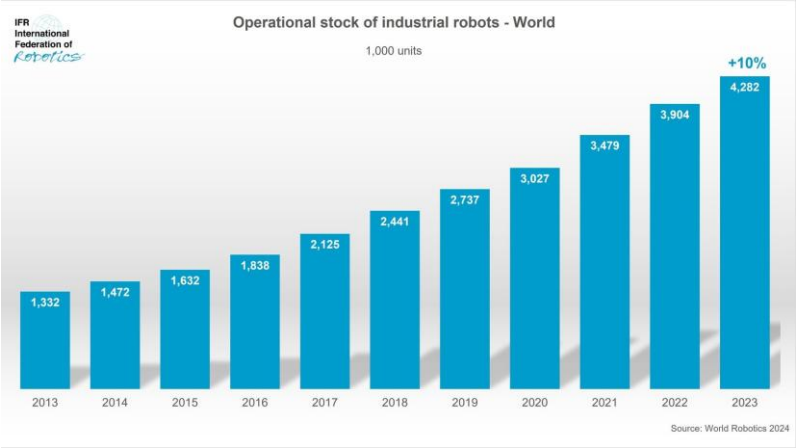


Figura 1.1: Evolución del uso de robots industriales

Con un punto de vista más global, el ámbito de aplicación de esta tecnología es bastante amplio. Su campo de aplicación más destacado es el de la industria, proporcionando la automatización en partes de la cadena de montaje o manufactura. Gracias a ello, se ha visto un gran incremento en la capacidad de producción, manteniendo o incluso mejorando, la precisión y la calidad de los productos [2].

Por otro lado, cabe destacar su implementación en el campo de la agricultura y la medicina. Siendo el primero utilizado para la recolección y evaluación de la calidad de las frutas, como se puede observar en el robot BERRY [3]. En cuanto a la medicina, se hace uso para la identificación de enfermedades y su tratamiento, agilizando los diagnósticos y realizando intervenciones quirúrgicas de alta precisión [4]. El ejemplo más notorio de ello es el sistema "da Vinci", capaz de realizar intervenciones mínimamente invasivas, a través de brazos robóticos que perfeccionan los movimientos del cirujano, evitando el temblor de las manos [5].

Estos son algunos ejemplos que demuestran la importancia del desarrollo robótico para diversas áreas profesionales. Cabe resaltar que, aparte de sus beneficios en términos operativos, también destaca por desempeñar un papel fundamental en la seguridad laboral. La implementación de estas máquinas en entornos hostiles minimiza la exposición a situaciones de peligro a las que los trabajadores pueden enfrentarse, reduciendo así la posibilidad de accidentes.

A través de este proyecto se quiere buscar una alternativa a los sistemas ya creados implementando un control robótico remoto. Este permite actuar de forma eficiente sobre el entorno físico, aumentando la seguridad laboral en diversas áreas.

1.2. Objetivos

El objetivo de este Trabajo de Fin de Grado es diseñar una aplicación web capaz de controlar y supervisar de forma remota un brazo robótico. Durante la duración del proyecto se implementan sistemas basados en microprocesadores, sistemas de comunicación y base de datos con la finalidad de integrar la parte hardware con el software. Todo ello, con la intención de desarrollar una interfaz de control intuitiva y sencilla para el usuario final.

A continuación, se especifican los objetivos que se han tenido en cuenta durante el transcurso del proyecto:

- Desarrollar el movimiento de los motores del brazo robótico, creando acciones predefinidas y desplazamientos individuales de cada actuador. Para ello, se hace uso de un microprocesador ESP8266MOD, el cual envía señales de ancho de pulso para ejecutar las acciones a realizar.
- Configurar la conexión inalámbrica del microcontrolador para establecer comunicación con el servidor web. Con este fin, se desarrolla un sistema que permite al usuario introducir las credenciales de la red deseada.

- Desplegar un servidor usando NodeJS que aloje la aplicación web, permitiendo el control y monitoreo del brazo robótico. Siempre teniendo en cuenta el diseño de la interfaz web para que sea intuitiva y fácil de usar.
- Establecer la comunicación entre el dispositivo hardware, la base de datos y el servidor a través del protocolo MQTT, garantizando el intercambio de mensajes y manteniendo una conexión persistente.
- Diseñar una base de datos para el almacenamiento de información relativa al estado del brazo robótico y el servidor web. Una vez establecida, visualizar los datos a través de un panel de control.
- Realizar pruebas al sistema para observar posibles fallos y mejorar su funcionalidad. Documentando cada parte realizada para su posterior descripción en la memoria técnica.

1.3. Planificación temporal

En esta sección se describen las diferentes etapas del desarrollo del proyecto, desde la propuesta inicial hasta el resultado final.

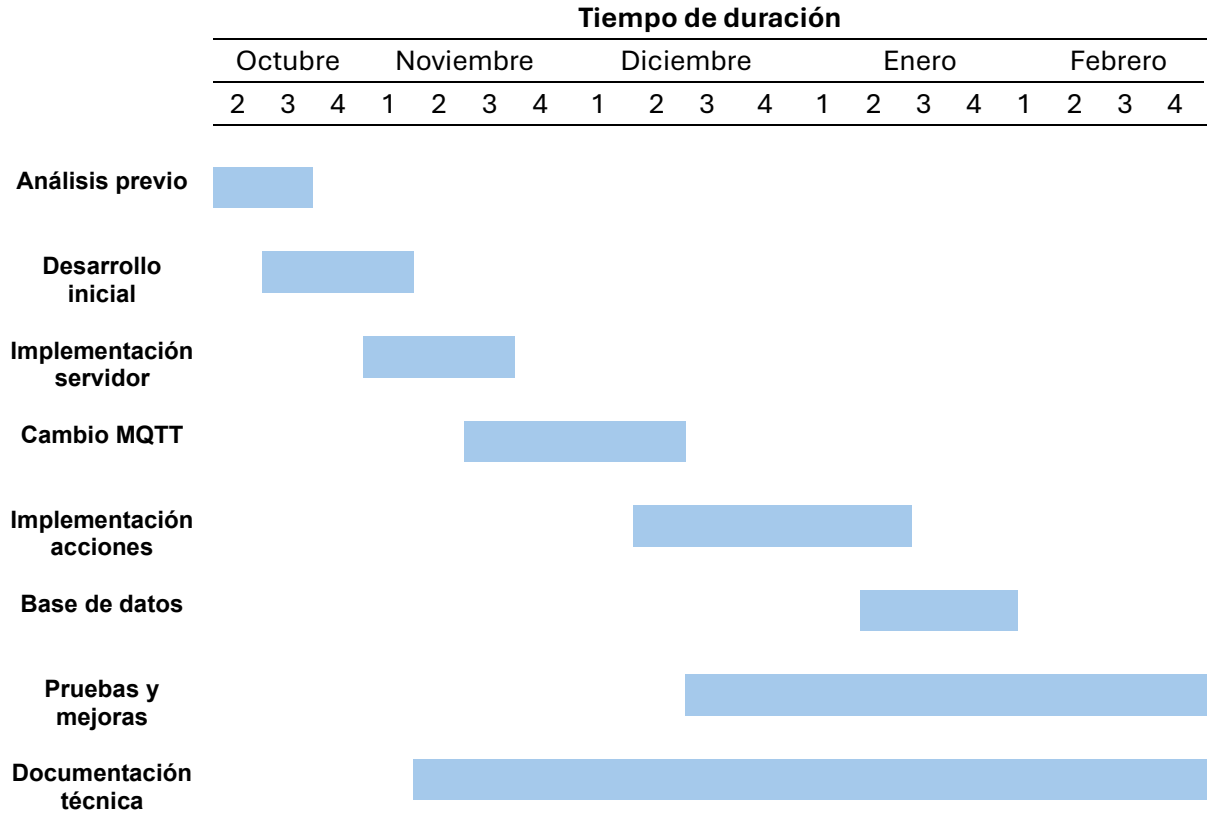


Tabla 1.1: Diagrama de Gantt del proyecto

1. **Análisis previo:** Durante las dos primeras semanas del proyecto se ha realizado un análisis de los componentes físicos del proyecto. El objetivo de dicha etapa es entender el funcionamiento de los servomotores. Por ello, se ha estudiado el rango de actuación de los motores, el sistema de comunicación y la conexión a los pines del microprocesador.
2. **Desarrollo inicial:** Tras el análisis previo, se realiza un código inicial para controlar los servomotores del brazo robótico. Se desarrolla una plataforma web inicial, la cual era alojada en la memoria interna del microprocesador y podía ser accedida a través de un punto de acceso. La comunicación entre ambos segmentos se realizaba mediante el protocolo HTTP, con un servidor RESTful ubicado en el controlador.
3. **Implementación del servidor:** Una vez comprobada la interacción entre la aplicación inicial y el brazo robótico, se desplazó la página a un servidor en NodeJS. Gracias esto, la aplicación es alojada en un servidor diferente al del microcontrolador y se desarrollan las funciones del back-end para actuar como intermediario entre controlador y la interfaz web.
4. **Cambio a MQTT:** Tras comprobar el correcto funcionamiento del sistema, se cambia el protocolo de comunicación a MQTT. El objetivo de este cambio es mantener una conexión persistente frente a una frecuencia alta de envío y la implementación de la base de datos.
5. **Implementación de acciones:** En esta fase se programan las funciones predefinidas que el usuario puede ejecutar a través de la página web. Además, se desarrolla la visualización de la interfaz web, permitiendo observar el estado del brazo robótico.
6. **Base de datos:** Para almacenar toda la información relevante del sistema se implementa una base de datos. En este periodo se desarrolla un programa capacitado para almacenar la información en InfluxDB, así como la configuración de Grafana para visualizar todos los datos almacenados.
7. **Pruebas y mejoras:** Durante la parte final del proyecto se han ido realizando pruebas sobre el sistema final. A través de estas, se ha podido visualizar los posibles errores generados con la finalidad de ir mejorando el sistema.
8. **Documentación técnica:** A lo largo de la mayoría del proyecto se ha ido documentando todos los hitos conseguidos. A través de ello, se ha generado la descripción técnica.

2. Estado del arte

2.1. Historia de la robótica

Pese a la visión actual de que la robótica es considerada como una ciencia moderna, sus orígenes se remontan a tiempos antiguos. Desde la edad antigua, los humanos soñaban con crear máquinas con apariencia humana o animal, con capacidad de tener conciencia y actuar de manera autónoma. Consideraban a estas máquinas como una buena opción para realizar tareas forzosas, evitando así, que las pudieran desarrollar los seres humanos. En este apartado se destacan los hitos más relevantes de la historia de la robótica.

Durante la segunda mitad del siglo V a.C., el filósofo pitagórico Arquitas de Tarento, diseño "la paloma voladora". Siendo uno de los primeros estudios sobre el vuelo de las aves, su premisa era hacer volar un cuerpo de madera con estructura similar a la de una paloma. Con una forma cilíndrica y con alas posicionadas a los lados, el ave era impulsada por vapor suministrado por una caldera. Gracias a su mecanismo interno, esta maqueta podía realizar vuelos cortos de manera autónoma, por lo que se considera el primer robot de la historia [6].

Uno de los primeros autómatas documentados es el creado por Leonardo Da Vinci en el año 1495. Esta máquina con forma de caballero podía emitir sonidos y realizar movimientos con las extremidades. Para realizar dichos movimientos contaba con un mecanismo el cual era accionado a través de un sistema de manivelas para desplazar los brazos y piernas. En la zona del pecho contaba con un pequeño tambor el cual permitía realizar sonidos de manera independiente. Dicho robot tenía como objetivo el sorprender a los invitados de la casa Sforza de Milán, los cuales tenían gran apego con Leonardo Da Vinci [7].

Continuando con uno de los grandes inventores, Jacques de Vaucanson es considerado como el primer ingeniero en especializarse en la creación de autómatas. Entre sus inventos destaca su primer autómata, "El Flautista" y "El pato con aparato digestivo". Este último fue creado en 1739, estaba fabricado en cobre a tamaño real y contaba con más de cuatrocientas piezas móviles simulando el aparato digestivo del animal. Gracias a este mecanismo, permitía a la máquina beber, comer grano y digerirlo. Además, a través de un sistema que imitaba a la anatomía de un pato, permitía al autómata batir sus alas, así como caminar y nadar sobre el agua [8].

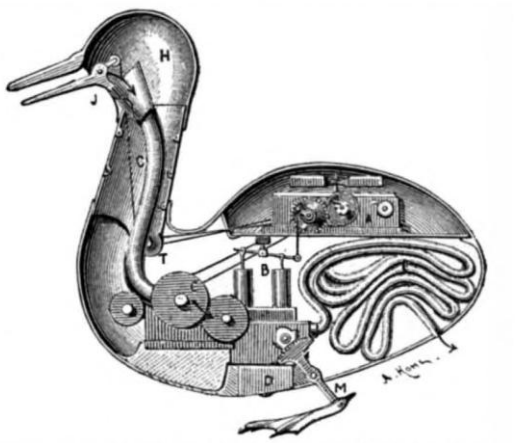


Figura 2.1: Representación del pato de Vaucanson

Años después con la llegada de la Primera Revolución Industrial, se desarrollaron diversos sistemas que dieron lugar a la automatización en el sector industrial. La implementación de una nueva fuente de energía, la máquina de vapor, facilitó el trabajo en diferentes áreas sentando las primeras bases de la industria moderna. A partir del año 1870, durante la Segunda Revolución Industrial, se desarrollaron innovaciones claves para el devenir de la robótica. En esta época se crearon sistemas alimentados por la electricidad y el petróleo a través del motor de combustión, dando pie al desarrollo de máquinas más complejas [9].

No obstante, no fue hasta el año 1921 cuando el término "robot" apareció tal y como lo conocemos hoy [10]. En este año el escritor checo Karel Čapek, escribió la obra de teatro R.U.R, donde se describía el uso de máquinas con apariencia humanoide que actuaban como siervos de los seres humanos. Debido al carácter de los autómatas, el escritor utilizó el termino robot el cual deriva del checo "robota", cuyo significado es trabajo forzoso [11]. A través de esta obra y las posteriores obras literarias del escritor Isaac Asimov, la percepción sobre estas máquinas cambiaría.

Tras años de innovaciones, en 1954 fue creado el primer robot industrial, Unimate. Diseñado por George Devol, era un brazo robótico programable el cual estaba capacitado para desplazar objetos de manera autónoma. Cinco años después, junto con la ayuda de Joseph Engelberger, este prototipo fue instalado en la cadena de montaje de la planta de fundición de General Motors en Nueva Jersey. La visión de Engelberger era que este prototipo fuese capaz de desempeñar tareas que suponían un riesgo para los trabajadores, evitando así la mayoría de los accidentes que se producían en esa época. El éxito de Unimate marcaría una nueva etapa en el uso de la robótica en el sector de la industria, impulsando la creación de nuevas máquinas para su implementación en diversas áreas de producción [12].

Figura 2.2: Robot industrial Unimate

También en la segunda mitad del siglo XX se lograron importantes avances tecnológicos. Durante los años 1966 y 1972, el Centro de Inteligencia Artificial del Instituto de Investigación de Stanford desarrolló Shakey. Equipado con sensores y cámaras fue el primer robot capaz de percibir y entender su entorno. A través, de estos sensores era capaz de definir rutas para desplazarse en espacios cerrados. Este proyecto supuso un gran avance tanto en la robótica moderna como en la inteligencia artificial, ya que combina la parte física con el procesamiento de datos a través de la visión artificial [13].

Durante la misma década el grupo de investigación del profesor Hiroshi Makino en la universidad de Yamanashi creó SCARA. Un brazo robótico con cuatro grados de libertad capacitado para desarrollar movimientos horizontales, mientras que se mantiene rígido sobre el eje vertical. Gracias a su sencilla composición proporciona movimientos a alta velocidad manteniendo una gran precisión [14].

En el siglo XXI se han conseguido hitos relevantes para el desarrollo de la robótica. Su aplicación se ha extendido en numerosos campos de la vida cotidiana y profesional. Cabe recalcar el robot quirúrgico da Vinci, el cual supuso una revolución a la hora de realizar intervenciones mínimamente invasivas [15]. Por otro lado, destaca el impacto de estos sistemas en la investigación espacial. Un de ello ejemplo es Canadarm2, un brazo robótico instalado en la ISS para desplazar objetos o personas. Este dispositivo destaca ya que puede ser controlado remotamente desde la estación espacial o por el equipo de tierra [16].

Estos son algunos de los hitos más relevantes relacionados con la historia de este campo. Actualmente, la robótica experimenta un crecimiento enorme gracias a la inteligencia artificial y la mejora del hardware. Impulsando su uso en la mayoría de los aspectos cotidianos, mediante sistemas mucho más prácticos y eficientes.

2.2. Brazo robótico

Un brazo robótico es un mecanismo programable diseñado para imitar la estructura y funcionalidad de un brazo humano. Por lo general, la creación de estos mecanismos presenta similitudes con la anatomía de la extremidad humana, teniendo articulaciones que se asemejan a la de los hombros, codos y muñecas, permitiendo la ejecución de movimientos. Cada una de estas articulaciones está unida por eslabones solidos que dan forma al conjunto del sistema.

Estos dispositivos están compuestos por varias partes para su funcionamiento [17]:

- **Controlador:** Es un microordenador programado para realizar cálculos, procesamientos de información y los movimientos del dispositivo. En el caso de este proyecto se ha optado por el microprocesador ESP8266MOD de Espressif el cual ha sido programado a través de Arduino.
- **Actuadores:** Parte mecánica encargada de generar la fuerza necesaria para el desarrollo de los movimientos del robot. Según la capacidad de carga necesaria, se hacen uso de motores eléctricos, hidráulicos o neumáticos.
- **Manipulador:** Representa toda la estructura mecánica del brazo robótico, compuesta por eslabones rígidos unidos a las articulaciones. Su diseño marca el propósito final del robot, ya que define el rango de movimiento.
- **Articulaciones:** Son los ejes de movimiento del manipulador, permitiendo realizar movimientos lineares o angulares. Cada articulación añade un grado de libertad al brazo.
- **Muñeca:** Es la sección encargada de la movilidad del efector final del brazo robótico. Su diseño puede tener diferentes grados de libertad, permitiendo desplazamientos como desviación, elevación y giro.
- **Mano robótica o efector final:** Parte final del manipulador, responsable de la funcionalidad mecánica final. Generalmente, usada para agarrar y desplazar objetos de un lado a otro. Pueden adoptar la forma de una pinza o replicar la anatomía de una mano humana.

Al hacer un estudio sobre la robótica, es fundamental remarcar la importancia del concepto de grados de libertad. Los grados de libertad (DOF), definen el número de movimientos independientes que un actuador puede realizar en un espacio tridimensional. Este valor representa, en gran parte, la flexibilidad del manipulador; indicando que cuanto mayor sea, más hábil es el brazo [18].

Cada grado de libertad está asociado a una articulación, que se distingue entre la capacidad de realizar movimientos lineales o rotativos. Por ello, la clasificación entre los diferentes brazos robóticos suele realizarse en función de los grados de libertad y el tipo de movimiento que ejecuta cada actuador. A continuación, se destacan los distintitos tipos de brazos robóticos:

- **Cartesiano:** Este tipo de robot está normalmente formado por tres actuadores lineales perpendiculares entre sí, permitiendo desplazamientos en los ejes X, Y o Z. Cada actuador se mueve solo sobre uno de los ejes, haciéndolo un sistema sencillo y rígido [19].
- **Cilíndrico:** El robot cilíndrico tiene un sistema de coordenadas cilíndricas, conformado por una articulación rotacional y dos lineales. La base del robot produce un desplazamiento rotacional alrededor de su propio eje. Mientras que, los dos ejes lineales controlan el movimiento vertical y radial [20].
- **Esférica o Polar:** Similar al cilíndrico, con la diferencia de que cambia el eje lineal vertical por un eje rotativo. Esta configuración permite un movimiento de inclinación habilitando desplazamientos dentro de un rango esférico [21].
- **Articulado:** Son los brazos robóticos con mayor abanico de campo de uso. Generalmente compuestos por cuatro o más articulaciones rotacionales, dotándolo de una mayor flexibilidad.
- **SCARA:** Este robot se caracteriza por realizar movimientos precisos en los ejes X-Y del plano cartesiano. Contando con cuatro grados de libertad, se reduce a realizar maniobras horizontales, manteniéndolo rígido sobre el eje Z y rotando sobre él [22].
- **Paralela o Delta:** Característicos por su gran velocidad y aceleración, consta de varias articulaciones conectadas a una base central con una disposición similar a un paralelogramo. El efector final está suspendido por debajo de la base y se encuentra conectado mediante barras rígidas a los motores [23].

El objetivo de estos mecanismos es, principalmente, el desplazamiento y agarre de diferentes tipos de objetos. Debido a esto, tienen capacidad de actuar de manera autónoma o manual dependiendo del entorno de trabajo. En situaciones que requieran de acciones repetitivas y constantes, se hace uso de un sistema autónomo. En cambio, cuando es necesario un control preciso del mecanismo, se opta por el manejo a través de interfaces físicas o digitales, permitiendo un control remoto sobre la máquina.

3. Recursos

Esta sección está destinada a detallar los recursos necesarios para alcanzar el objetivo final de este Trabajo de Fin de Grado. En ella se describen los componentes físicos y software elegidos para su desarrollo, explicando la función de cada uno y su implementación.

3.1. Hardware

Los componentes físicos requeridos para el despliegue del proyecto están estrictamente relacionados con el funcionamiento del brazo robótico. Entre estos componentes se incluyen todos los elementos estructurales del brazo, la unidad de control y la alimentación del sistema.

3.1.1. Brazo robótico

Para este proyecto se ha utilizado un brazo robot de 6 grados de libertad (6DOF) extendido [24]. Forma parte de la categoría de brazo robótico articulado con un total de seis ejes. Cada articulación es accionada por un servomotor eléctrico que permite realizar movimientos rotacionales. Todos estos motores están dispuestos en posiciones distintas, permitiendo una función específica para cada uno.

Comenzando desde la parte inferior del brazo, el actuador de la base permite una rotación de la totalidad del manipulador alrededor suyo. Los dos servomotores siguientes, se encargan del movimiento de inclinación del brazo, facilitando el descenso y elevación. Mientras que, los tres últimos se encargan de accionar el efector final, desplazando la posición de la pinza y facilitando su apertura o cierre.

Figura 3.1: Brazo robótico 6DOF

Debido a que se trata del modelo de brazo robótico extendido, se incluyen dos servomotores con una mayor capacidad de carga. El modelo básico de los servomotores, MG995R, se posicionan en la base y en los tres segmentos del efector final. Mientras que, los dos actuadores restantes son modificados por los servomotores DS3115MG, situados en las articulaciones que permiten la inclinación del brazo.

3.1.2. Servomotores

Los desplazamientos que realiza el brazo robótico son creados mediante el uso de servomotores. Como se ha comentado posteriormente, el brazo robótico elegido cuenta con seis actuadores que permiten accionar cada una de sus articulaciones. Estos destacan por ser motores eléctricos que proporcionan movimientos rotacionales, asegurando una gran precisión y velocidad.

En cuanto a las especificaciones de los servomotores, se cuentan con los actuadores MG996R [25] y DS3115MG [26]. Estos se caracterizan por su tamaño compacto y ligero en relación con la potencia que generan. Ambos constan de tres pines referentes a la entrada de alimentación, tierra y el ancho de pulso. Pese a que presentan similitudes, existen importantes diferencias, en la que destaca el par de parada. En la siguiente tabla se detallan las características y diferencias de ambos actuadores:

Características	MG996R	DS3115MG
Peso (g)	55	60
Dimensiones (mm)	40.7 x 19.7 x 42.9	40x20x40.5
Voltaje de operación (V)	4.8 – 7.2	4.8 – 6.5
Velocidad de operación mínima (seg / 60 grados)	0.17 (4.8V sin carga)	0.15 (4.8V sin carga)
Velocidad de operación máxima (seg / 60 grados)	0.14 (6V sin carga)	0.13 (6.5V sin carga)
Par de parada (Kg)	9.4 (@4.8V) / 11 (@6V)	14.5 (@4.8V) / 17.3 (@6.5V)
Frecuencia de pulso (Hz)	50 (20ms)	50 (20ms)
Ancho de pulso (ms)	0,5 (0º) ~ 2,5 (180º)	0,5 (0º) ~ 2,5 (180º)

Tabla 3.1: Características de los servomotores

El funcionamiento de estos motores se fundamenta en el envío de señales de modulación por ancho de pulso (PWM). Esta técnica se basa en una señal periódica que modifica el ciclo de trabajo para regular la cantidad de energía enviada. Con ciclo de trabajo se refiere al tiempo que la señal está activa con respecto al ciclo completo. A continuación, se anota la expresión matemática del ciclo de trabajo, siendo TON y TOFF el tiempo que la señal está activa e inactiva, respectivamente.

$$Duty\ Cycle\ = \frac{TON}{TON + TOFF}$$

En el caso de estos servomotores, el ciclo de trabajo indica el movimiento a realizar. Para ambos actuadores, la señal consta de un periodo de 20 milisegundos, donde el ancho de pulso puede variar entre 500 y 2500 microsegundos. Siendo el valor mínimo el que posiciona el motor en el extremo de 0 grados, mientras que el mayor lo posiciona en el extremo de 180 grados. Gracias esto, el actuador es capaz de realizar rotaciones en un rango de 180 grados.

3.1.3. Microcontrolador

Para garantizar el control del brazo robótico y la comunicación con el resto del sistema, se ha escogido el microcontrolador ESP8266MOD. Este módulo de control está basado en la tecnología del chip ESP8266 desarrollado por Espressif [27]. Este proporciona la posibilidad de ser programado mediante diversas plataformas para implementar sus funciones. En el caso de este proyecto, la primera funcionalidad de la que se ha hecho uso es la conexión a redes inalámbricas mediante su módulo WiFi. Dicho módulo permite al microcontrolador actuar como cliente de una conexión en el modo estación (STA) o servir como un punto de acceso (AP).

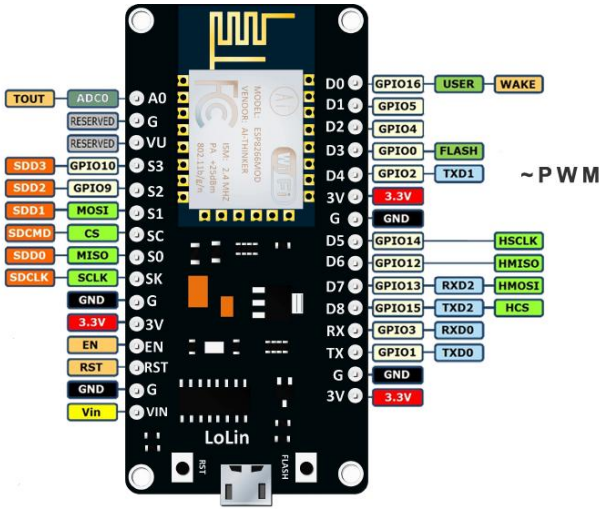


Figura 3.2: Referencia Pinout ESP8266MOD

El microcontrolador ESP8266MOD cuenta con diecisiete pines de entrada y salida (GPIO), facilitando el control y la comunicación con múltiples dispositivos de manera simultánea. Cada uno de estos conectores pueden ser usados para diferentes tareas, como la comunicación en serie mediante los protocolos I2C y SPI, el envío de señales de ancho de pulso, la conversión analógica digital y el control de dispositivos digitales, entre otras. Además, este dispositivo cuenta con dos pines de alimentación de 3.3V y otros dos para la toma de tierra.

Puesto que el brazo robótico cuenta con seis servomotores cuyo control debe ser simultaneo, se han requerido de seis pines de entrada/salida del módulo. Dichos conectores corresponden a los que abarcan desde D1 a D6, los cuales se pueden observar en la Figura 3.2. Su elección se ha realizado teniendo en cuenta que son capaces de enviar las señales de ancho de pulso para accionar los motores y no tienen otro propósito definido.

3.1.4. Fuente de alimentación y placa de circuito

Para el correcto funcionamiento del brazo robótico es necesaria la implementación de una alimentación externa. Esta debe permitir el desplazamiento de cada uno de los motores de manera continuada pese a soportar una gran carga. Para ello, se debe escoger una fuente de alimentación que suministre suficiente voltaje para entrar en los valores de tensión de funcionamiento de ambos tipos de actuadores.

Atendiendo a las especificaciones de los motores, se puede deducir que la fuente de alimentación a usar debe suministrar entre los cinco y siete voltios. También es importante entender que cuanto mayor sea su voltaje de entrada mayor es el par de parada, lo que implica que puede soportar mayor peso. En el caso de este proyecto se utiliza una fuente de alimentación que suministra cinco voltios, siendo suficiente para el desarrollo de las diferentes tareas diseñadas para el brazo robótico.

Las conexiones entre los distintos componentes físicos deben realizarse a través de una placa que permita su correcto funcionamiento. Tanto el microcontrolador, como la fuente de alimentación y los pines de los servomotores necesitan una plataforma en la cual se suministre la energía y señales necesarias. Para este proyecto se hace uso de una placa de pruebas con el espacio suficiente para realizar cada una de las interconexiones mediante el cableado. La protoboard usada proporciona un entorno flexible el cual permite la modificación de cada una de las conexiones, al no hacer uso de soldaduras previas.

3.2. Software

Durante el transcurso de este proyecto se ha requerido del uso de varias aplicaciones para la implementación del sistema. Estas plataformas abarcan desde los entornos de programación utilizados, hasta la implementación del servidor y la base de datos. En esta sección se enumeran cada uno de los componentes software, definiendo su propósito:

- **Arduino IDE:** Entorno de desarrollo en el que se escribe todo el código de control del hardware. Elegido por su compatibilidad con el microcontrolador, emplea un lenguaje de programación basado en C++. Arduino actúa como compilador del código escrito para su posterior carga en el microcontrolador. Esta parte del proyecto se encarga de recibir mensajes desde el servidor web, procesarlos y enviar las acciones al robot. También es usado para la conexión con la red inalámbrica y el manejo de intercambio de mensajes con la base de datos y el servidor web.
- **NodeJS:** Entorno de ejecución de JavaScript capacitado para desplegar el servidor web. A través de esta herramienta se configura la comunicación entre el microcontrolador y la aplicación web. Además, esta plataforma aloja la aplicación web en un puerto determinado y gestiona las acciones definidas para el back-end.
- **Mosquitto:** Plataforma que actúa como servidor MQTT, permitiendo la comunicación entre las partes del sistema. Este broker facilita la conexión de los clientes para poder suscribirse o publicar mensajes. Gracias a esta herramienta, se permite la comunicación entre el microcontrolador, servidor web y base de datos.
- **InfluxDB:** Base de datos de series temporales que almacena todos los datos relacionados con los ángulos del brazo, peticiones de la aplicación y estado del servidor web.
- **Grafana:** Herramienta que permite visualizar la información almacenada en la base de datos a través de consultas. Ofrece un panel de monitoreo que permite visualizar gráficamente todos los datos relevantes del sistema, facilitando una visión general del sistema a simple vista.
- **Docker:** Permite desplegar las herramientas de la base de datos y la visualización en Grafana a través de contenedores aislados. Esto facilita la configuración y ejecución de ambos entornos, así como su integración.

4. Arquitectura del sistema

Con el fin de alcanzar el objetivo final de este proyecto se ha desplegado un sistema formado por el control del brazo robótico, un servidor web y la plataforma de almacenamiento y visualización de datos. Esta arquitectura permite el control y la monitorización del brazo robótico a través de un buscador en un dispositivo móvil o computadora conectado a la misma red inalámbrica que el resto del sistema.

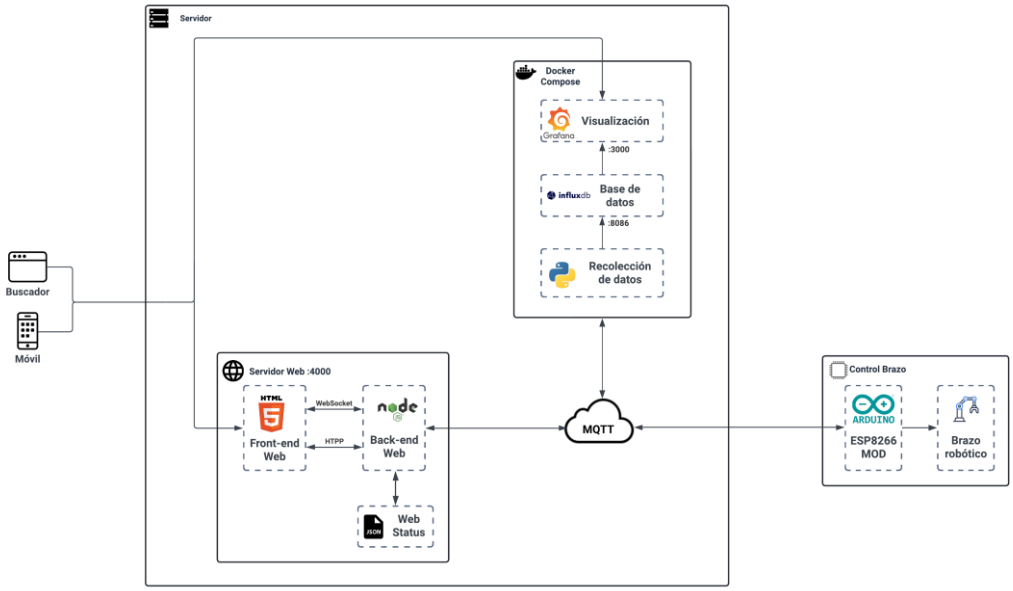


Figura 4.1: Diagrama de arquitectura del sistema

- **Control del brazo robótico:** Parte hardware del proyecto, formada por el microcontrolador ESP8266MOD y el brazo robótico. Su función es accionar el brazo y facilitar el intercambio de mensajes con la capa de usuario.
- **Servidor web:** Servidor alojado en el puerto 4000 de la máquina, encargado de alojar tanto front-end como el back-end de la página web. La interacción entre ambas partes se establece mediante los protocolos HTTP y WebSocket, permitiendo el envío de peticiones puntuales y la actualización constante de la interfaz web.
  - **Back-end Web:** Implementado con NodeJS, opera como intermediario entre la capa de interfaz de usuario y el brazo robótico.
  - **Web Status:** Archivo JSON que almacena el estado de la interfaz gráfica de la página web.
  - **Front-end Web:** Capa de interfaz web formada por botones y controles deslizantes, facilitando el control y monitorización del brazo.

- **Almacenamiento y visualización de datos:** Este segmento del sistema es el encargado de recolectar los datos relacionados con el robot y el servidor web para su posterior monitorización.
  - **Recolección de datos:** Programa desarrollado en Python, actúa como middleware con el objetivo de escuchar los mensajes publicados en el broker y escribir su información en la base de datos.
  - **Base de datos:** Almacena en la plataforma InfluxDB alojada en el puerto 8086, los datos de ángulos y acciones realizadas por el brazo, así como las métricas de rendimiento del servidor web.
  - **Visualización:** Mediante la herramienta Grafana, se desarrolla un panel el cual está capacitado para mostrar todas las variables almacenadas. El usuario puede acceder a su interfaz web a través del puerto 3000.

Para garantizar la comunicación entre las partes del sistema se ha establecido un broker de Mosquitto. Este proporciona una comunicación bidireccional siguiendo un modelo de publicación y suscripción. Gracias a ello, los clientes conectados a la misma red inalámbrica pueden escribir y escuchar mensajes en diferentes temas. Los temas establecidos para el funcionamiento del sistema son:

- **esp8266/moveServo:** Tema en el que se publican las peticiones de la página web para mover un servomotor.
- **esp8266/predefinedMovement:** Se encarga de gestionar las solicitudes de la página web para ejecutar acciones predefinidas. Estas acciones son: bajar el brazo, cancelar la bajada, agarrar un objeto, soltarlo y reiniciar el brazo a su posición de origen. Esta consulta es realizada desde la página web, obteniendo como respuesta la posición final del brazo enviada por el microcontrolador
- **esp8266/servoAngles:** Maneja la solicitud que realiza la página web al ser iniciada con el objetivo de recibir la posición actual del brazo robótico. El microcontrolador la recibe y responde con los ángulos de los motores.
- **esp8266/startArm:** Tema al que publica el microcontrolador al ser iniciado indicando su arranque y la posición del brazo.
- **esp8266/angles:** El microcontrolador publica periódicamente la posición del brazo robótico para almacenarlo en la base de datos.
- **system/metrics:** En este tema el servidor web envía periódicamente las métricas de rendimiento del sistema para almacenarlas en la base de datos.

5. Descripción técnica del proyecto

5.1. Control del brazo robótico

En esta parte del sistema se ha desarrollado un programa en la plataforma Arduino IDE. La finalidad de este código es configurar el microcontrolador ESP8266MOD para poder manejar los servomotores del brazo robótico. Dado los requisitos del sistema, también se ha utilizado para el intercambio de mensajes MQTT con la página web y la base de datos.

A la hora de desarrollar el código se ha hecho uso de diversas librerías:

- **ESP8266WiFi.h:** Librería usada para manejar la conexión a redes inalámbricas, permitiendo actuar al microcontrolador en modo estación (STA) o punto de acceso (AP).
- **ESP8266WebServer.h:** Esta librería permite la creación de un servidor web en el microprocesador, facilitando el manejo de solicitudes HTTP. Su función en este proyecto es la configuración de la conexión con la red inalámbrica.
- **LittleFS.h:** Biblioteca que permite leer, escribir y borrar archivos en la memoria no volátil del microcontrolador.
- **ArduinoJson.h:** Librería que facilita la escritura y lectura de los archivos JSON utilizados para el intercambio de mensajes.
- **Servo.h:** Permite el manejo de servomotores, facilitando la escritura y lectura de los ángulos.
- **PubSubClient.h:** Esta librería maneja tanto la conexión con el broker, como el procesamiento del envío y recepción de mensajes del protocolo MQTT.

Clase WiFiHandler

Esta clase está diseñada para conectar el microcontrolador a la red inalámbrica deseada por el usuario final. Al iniciar el dispositivo es importante establecer una conexión a la red del sistema para permitir el intercambio de mensajes con cada una de las partes de su arquitectura. Por ello, al ser arrancado, intenta conectarse a la red cuyas credenciales permanecen guardadas en su memoria flash. Si no es capaz de establecer conexión o si no se encuentra ninguna credencial en la memoria, se despliega un punto de acceso. A través de este se permite el acceso a una página web en la que se pueden ingresar los datos de acceso de la red deseada.

A continuación, se desarrolla el flujo que sigue esta clase:

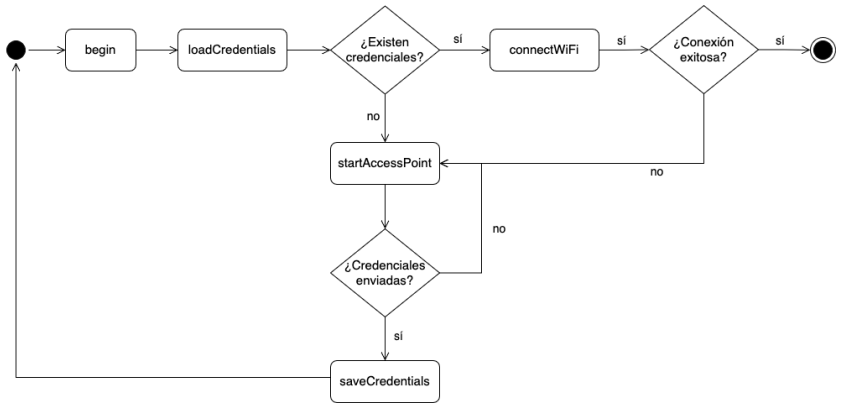


Figura 5.1: Diagrama de flujo de WifiHandler

- **begin**: Función de inicio encargada de ejecutar el flujo inicial. Este método comprueba que el sistema de control de archivos ha sido iniciado con éxito y llama a la función de carga de credenciales.

```
void WiFiHandler::begin() {
  if (!LittleFS.begin()) {
    Serial.println("Error al iniciar LittleFS");
    return;
  }
  Serial.println("Intentando conectar al WiFi...");
  loadCredentials();
}
```

Figura 5.2: Función begin de WiFiHandler

- **loadCredentials**: Maneja la lectura del archivo guardado en la memoria interna. Si detecta que el archivo contiene alguna clave de acceso intenta su conexión. Mientras que, si no se encuentra el archivo o ninguna credencial, inicia el punto de acceso.

```
void WiFiHandler::loadCredentials() {
  if (LittleFS.exists("/wifi.json")) {
    File wifiFile = LittleFS.open("/wifi.json", "r");
    if (wifiFile) {
      DynamicJsonDocument doc(512);
      DeserializationError error = deserializeJson(doc, wifiFile);
      wifiFile.close();
      if (error) {
        startAccessPoint();
        return;
      }
      if (!doc.containsKey("ssid") || !doc.containsKey("password")) {
        startAccessPoint();
        return;
      }
      const char* ssid = doc["ssid"];
      const char* password = doc["password"];
      connectWiFi(ssid, password);
    }
  } else {
    startAccessPoint();
  }
}
```

Figura 5.3: Función loadCredentials

- **connectWiFi**: Inicia la conexión a la red inalámbrica. Si la conexión es exitosa, se mantiene y finaliza el flujo; de lo contrario, se inicia el punto de acceso.

```
void WiFiHandler::connectWiFi(const char* WiFissid, const char* WiFiPassword) {
  WiFi.begin(WiFissid, WiFiPassword);
  unsigned long startTime = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - startTime < 10000) {
    Serial.println("Conectando al WiFi.....");
    delay(500);
  }
  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("Error al conectarse a la WiFi. Iniciando AP...");
    startAccessPoint();
  } else {
    Serial.println("Conectado a la WiFi");
  }
  delay(3000);
}
```

Figura 5.4: Función connectWiFi

- **startAccessPoint**: Inicia el modo AP, dando acceso a la web en la que se ingresan las claves de la red inalámbrica. Tras emitir las credenciales, las guarda y reinicia el microcontrolador para intentar establecer conexión.

```
void WiFiHandler::startAccessPoint() {
  Serial.println("Iniciando AP...");
  WiFi.softAP(_APssid, _APPassword);
  _server.onNotFound([this]() {
    _server.sendHeader("Location", "http://192.168.4.1/", true);
    _server.send(302, "text/plain", "");
  });

  _server.on("/", HTTP_GET, [this]() {
    _server.send(200, "text/html", loadWebPage());
  });
  _server.on("/config", HTTP_POST, [this]() {
    String ssid = _server.arg("ssid");
    String password = _server.arg("password");

    if (ssid.length() > 0 && password.length() > 0) {
      saveCredentials(ssid.c_str(), password.c_str());
      _server.send(200, "text/html", "Guardado, reiniciando...");
      delay(2000);
      ESP.restart();
    } else {
      _server.send(400, "text/html", "Parámetros inválidos.");
    }
  });
  _server.begin();
}
```

Figura 5.5: Función startAccesPoint

- **saveCredentials**: Almacena las claves de acceso suministradas por la página en la memoria flash. Para ello, convierte las variables en un documento JSON y las escribe a través de la librería LittleFS.

```
void WiFiHandler::saveCredentials(const char* WiFissid, const char* WiFiPassword) {
  if (WiFissid == nullptr || WiFiPassword == nullptr) {
    Serial.println("Error SSID o contraseña nulos");
    return;
  }

  DynamicJsonDocument doc(512);
  doc["ssid"] = WiFissid;
  doc["password"] = WiFiPassword;

  File wifiFile = LittleFS.open("/wifi.json", "w");
  if (wifiFile) {
    serializeJson(doc, wifiFile);
    wifiFile.close();
  }
}
```

Figura 5.6: Función saveCredentials

Control de los Servomotores

El control sobre los seis servomotores del brazo robótico se lleva a cabo mediante la librería mencionada anteriormente. Por ello, se inicializan un conjunto de seis servomotores, asociándolos con sus respectivos pines. A través de estos objetos, se permite la escritura de ángulos mediante el envío de una señal PWM.

Con el propósito de garantizar un desplazamiento suave y preciso, se ha establecido que todos los movimientos de los actuadores sean progresivos, evitando así la posibilidad de generar movimientos bruscos. Debido a ello, se hace uso de una función capaz de mover el actuador desde su posición actual hasta el ángulo objetivo de manera gradual mediante un bucle. Dentro de este bucle, se desplaza el ángulo del actuador de uno en uno desde el ángulo original hasta el final.

```
void moveServoAngle(int servoIndex, int angle) {
  int targetAngle = constrain(angle, 0, 180);
  int currentAngle = servos[servoIndex].read();

  int step = targetAngle > currentAngle ? 1 : -1;

  for (int pos = currentAngle; pos != targetAngle + step; pos += step) {
    servos[servoIndex].write(pos);
    delay(20);
  }
}
```

Figura 5.7: Función movimiento servomotores

Con la ayuda de esta función, se han establecidos algunas acciones predefinidas que el usuario puede activar desde la aplicación web. A continuación, se explican cada una de las secuencias implementadas:

- **resetServosToRef:** Desplaza el brazo robótico a la posición establecida como original.
- **bajarBrazo:** Inclina el brazo para proporcionar su descenso con la garra abierta para poder agarrar un objeto. Tras activar esta acción se abren dos posibilidades.
    - **agarrarObjeto:** Cierra la pinza para sujetar el objeto y eleva nuevamente el brazo.
    - **cancelarBajada:** Cancela la bajada solicitada y eleva el brazo a su posición original.
- **soltarObjeto:** Suelta el objeto previamente agarrado en la secuencia de bajada. Esta acción se encarga de inclinar el brazo robótico hasta abajo, para posteriormente abrir la garra y depositar el objeto en la posición deseada.

Manejo del protocolo MQTT

El intercambio de mensajes MQTT en el microprocesador tiene dos finalidades, la comunicación con el servidor web y el envío de ángulos a la base de datos. Al tratarse de un protocolo red basado en un modelo de publicación-suscripción, se puede diferenciar esta parte del código en función de estos dos mecanismos.

En cuanto a la suscripción, el microcontrolador espera recibir mensajes expedidos desde la página web habilitada para su control y supervisión. Por ello se suscribe a los temas *moveServo*, *predefinedMovements* y *servoAngles*. Temas en los que se solicita desplazar un actuador a un ángulo especifico, realizar una secuencia de movimientos y el envío de ángulos para ser visualizados en la interfaz web, respectivamente. Al ser iniciado el sistema, se realiza la suscripción a estos tres temas y espera activamente a la recepción de un mensaje.

Al recibir un mensaje, el microcontrolador debe estar preparado para determinar el tema del que proviene y actuar en consecuencia de ello. Con esta finalidad se desarrolla la función *mqttCallback*, la cual es ejecutada al interceptar un mensaje. Dicho método recoge tanto el contenido como el continente del mensaje para analizarlo. El primer paso que realiza es la conversión del payload a una cadena de caracteres, mediante un bucle que se ejecuta en función de su tamaño. Una vez completado el bucle, la cadena de caracteres es convertida a un objeto JSON para facilitar la obtención de los datos.

Con el mensaje en el formato correcto, el siguiente paso es definir el tema de origen de este mensaje. Teniendo en cuenta que cada uno de los topics tienen un propósito diferente, se filtra el flujo de ejecución a través de estos. Por ello, se define una estructura de control condicional que compara la cadena de caracteres recibida con cada uno de los temas suscritos. Dependiendo de ello actúa de diferente manera:

1. **Desplazamiento del servomotor:** Identifica las variables que contienen el número del motor y el ángulo objetivo. Tras esto, llama al método para desplazarlo gradualmente, siempre y cuando el índice del motor sea válido.
2. **Movimientos predefinidos:** Se comprueba el tipo de movimiento expedido por la página web. Dependiendo de la petición se ejecuta el método que permite realizar dicha acción. Una vez finalizada la secuencia de movimientos, se envía como repuesta la posición del brazo a la página web y a la base de datos.
3. **Petición de ángulos:** Se publica la posición actual del brazo para su visualización en la interfaz web.

```
void mqttCallback(char* topic, byte* payload, unsigned int length) {
  String message;
  for (unsigned int i = 0; i < length; i++) {
    message += (char)payload[i];
  }
  DynamicJsonDocument doc(512);
  DeserializationError error = deserializeJson(doc, message);
  if (error) {
    Serial.println("Error al procesar el JSON");
    Serial.println(error.c_str());
    return;
  }
  if (strcmp(topic, mqttTopics[0]) == 0) {
    int servoIndex = doc["servo"];
    int targetAngle = doc["angle"];
    if (servoIndex >= 0 && servoIndex < NUM_SERVOS) {
      moveServoAngle(servoIndex, targetAngle);
    } else {
      Serial.println("Indice del servo no valido");
    }
  } else if (strcmp(topic, mqttTopics[1]) == 0) {
    const char* movement = doc["movement"];
    if (strcmp(movement, "resetBrazo") == 0) {
      resetServosToRef();
    } else if (strcmp(movement, "bajarBrazo") == 0) {
      bajarBrazo();
    } else if (strcmp(movement, "agarrarBrazo") == 0) {
      agarrarObjeto();
    } else if (strcmp(movement, "cancelarBajada") == 0) {
      cancelarBajada();
    } else if (strcmp(movement, "soltarBrazo") == 0) {
      soltarObjeto();
    }
    publishMQTT("esp8266/predefinedMovement/response",movement);
    publishMQTT("esp8266/angles","anglesDB");
  } else if (strcmp(topic, mqttTopics[2]) == 0) {
    publishMQTT("esp8266/servoAngles/response","servoAngles");
  }
}
```

Figura 5.8: Manejo protocolo MQTT del microcontrolador

Por otro lado, las publicaciones que realiza el microcontrolador tienen como receptor final el servidor web y la base de datos. En cuanto a la página web, se envía la posición del brazo como respuesta a las peticiones y al ser iniciado el microprocesador. Mientras que, a la base de datos se le envía los ángulos de cada uno de los motores al finalizar un movimiento predefinido y de forma periódica en un intervalo de treinta segundos.

Con el propósito de realizar estos envíos, se crea la función *publishMQTT* la cual se encarga de la publicación de mensajes en el broker MQTT. Este método recibe como parámetros tanto el tema como los datos a publicar. A través de estos datos, se genera un documento JSON en el cual se especifica la acción y la posición del brazo. Finalmente, este documento es publicado en el tema indicado con éxito.

```
void publishMQTT(const char* topic, String action) {
  DynamicJsonDocument doc(512);
  doc["movement"] = action;
  JsonArray angles = doc.createNestedArray("angles");
  for (int index = 0; index < NUM_SERVOS; index++) {
    angles.add(servos[index].read());
  }
  String message;
  serializeJson(doc, message);
  if (mqttClient.publish(topic, message.c_str())) {
    Serial.println("Mensaje publicado correctamente:");
    Serial.println(message);
  } else {
    Serial.println("Error al publicar el mensaje");
  }
}
```

Figura 5.9: Publicación MQTT ESP8266

Funciones setup y loop

Una vez iniciado el microcontrolador, se ejecuta la función de inicialización *setup*. Esta se encarga de la configuración de las distintas partes del sistema. Primero se inicia la conexión con la red inalámbrica mediante la clase *WiFiHandler*. Si la conexión resulta exitosa, se enlazan las variables de cada uno de los seis servomotores a su respectivo pin y se desplaza todo el brazo a su posición definida como original. El último paso de este método es la conexión y configuración frente a los eventos de la plataforma MQTT. Tras confirmar la conectividad con dicha plataforma, el microcontrolador envía un mensaje comunicando su inicio.

```
void setup() {
  Serial.begin(9600);
  WiFiHandler.begin();
  if (WiFiHandler.isWiFiConnected()) {
    for (int servoIndex = 0; servoIndex < NUM_SERVOS; servoIndex++) {
      servos[servoIndex].attach(servoPins[servoIndex], MIN_PULSE_WIDTH, MAX_PULSE_WIDTH);
      delay(50);
      servos[servoIndex].write(servoPosRef[servoIndex]);
      delay(50);
    }
    mqttClient.setServer(mqttServer, mqttPort);
    mqttClient.setCallback(mqttCallback);
    connectMQTT();
    publishMQTT("esp8266/startArm", "Arm Started");
  }
}
```

Figura 5.10: Función de inicialización ESP8266

Al finalizar la función de inicialización, se ejecuta el bucle principal del programa. Este segmento es el encargado de manejar la conexión con la red inalámbrica y las peticiones recibidas desde el punto de acceso para obtener las credenciales WiFi. Si existe comunicación con la red inalámbrica, ejecuta la función *mqttClient.loop*, la cual se encarga de procesar los mensajes recibidos y mantener conexión con el broker. También, se procede a detectar si ya han pasado los treinta segundos para enviar la posición a la base de datos. Por último, en el caso de que haya una pérdida de conexión con la red WiFi, se reinicia el sistema para ejecutar *WiFiHandler*.

```
void loop() {
  WiFiHandler.handleClient();
  if (WiFiHandler.isWiFiConnected()) {
    if (!mqttClient.connected()) {
      connectMQTT();
    }
    mqttClient.loop();
    if (millis() - lastSendTime >= SEND_INTERVAL) {
      publishMQTT("esp8266/angles","anglesDB");
      lastSendTime = millis();
    }
  } else if (WiFiHandler.getmodeWiFi() == 1){
    ESP.restart();
  }
}
```

Figura 5.11: Función bucle ESP8266

5.2. Servidor web

El objetivo de este segmento del sistema es desarrollar la plataforma para poder interactuar con el hardware desplegado. Por ello, se diseña una aplicación web la cual está capacitada para accionar y monitorizar las diferentes partes del brazo. Dicha página se aloja en un servidor, el cual se divide entre la parte del front-end y del back-end, que se comunican mediante los protocolos HTTP y WebSocket.

El protocolo de transferencia de hipertexto es usado para el intercambio de consultas con el microprocesador para accionar el brazo. Su ejecución se establece a través de los botones y barras deslizantes de la página web. Por otro lado, la tecnología WebSocket se encarga de la comunicación simultánea para actualizar la interfaz web conforme al estado del brazo robótico. El uso de esta tecnología está pensado para generar un entorno colaborativo, permitiendo ver el accionamiento de las funcionalidades del robot desde varios dispositivos al mismo tiempo.

5.2.1. Desarrollo de la página web

Durante el desarrollo de la página web se ha buscado una interfaz sencilla y accesible. Consta de varios botones dispuestos en la parte superior e inferior de la pantalla capacitados para activar las secuencias de movimientos. Mientras que, en el centro se conforman seis barras deslizantes que ejecutan el control de cada uno de los servomotores. Para conseguir el funcionamiento de esta plataforma se han creado los archivos *index.html*, *style.css* y *script.js*, formando la base del front-end.

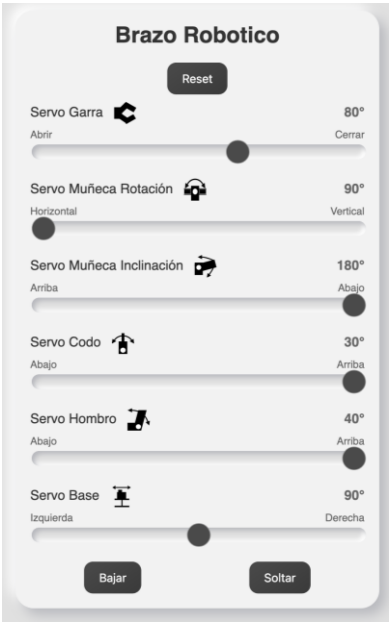


Figura 5.12: Interfaz web inicial

Diseño de la página web

Para el diseño de la interfaz web se utiliza el código *style.css*. En este código se ajustan todas las variables de diseño de la web, especificando el tamaño, el color y la fuente de cada uno de los elementos de la página. Como se puede observar en la Figura 5.12, los componentes están agrupados en un contenedor blanco que proporciona una apariencia organizada. Dentro de este contendor, se disponen los controles alineados verticalmente, con las barras y botones siguiendo la misma fuente y patrón de color.

En cuanto a los sliders, siguen un orden de abajo hacia arriba correspondiendo con la disposición de los motores en el brazo robótico. Para facilitar la relación entre cada control con su respectivo actuador, se le dan nombres que los compara con la anatomía del brazo humano. Además, se proporciona un icono que ilustra la posición y función de cada motor.

Por otro lado, en la parte superior se encuentra el botón "Reset", el cual desplaza el brazo a su posición original. Mientras que, en la parte inferior se encuentran los botones que permiten recoger y soltar un objeto. Cabe recalcar que, los botones de la parte inferior de la página están agrupados de diferente manera. Esto se debe a que las funciones de "Agarrar" y "Cancelar" solo pueden ser ejecutadas después de activar la bajada. Por ello, ambos controles son iniciados como oculto hasta que es pulsado el botón "Bajar". En la siguiente figura se puede observar que el botón de bajada es ocultado tras ser pulsado, permitiendo accionar la sujeción de un objeto o la cancelación de la bajada.

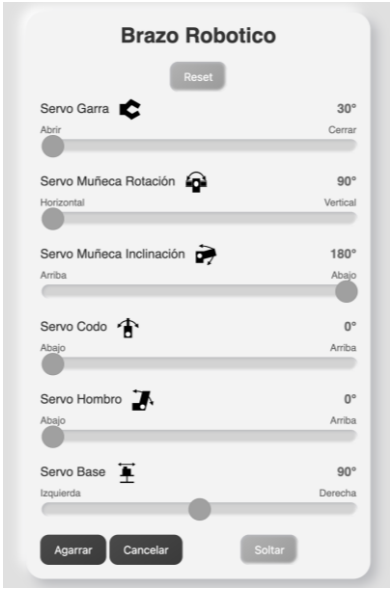


Figura 5.13: Interfaz tras finalizar bajada

Otra parte del diseño de esta interfaz es la habilitación e inhabilitación de las barras y los botones. Cuando un control permanece deshabilitado se impide la actuación sobre ellos y cambia visualmente a un color grisáceo. Este estado permite observar que el brazo robótico está en movimiento al haber pulsado uno de los botones. Mediante este mecanismo se evita la posibilidad de solicitar alguna acción durante una secuencia de movimientos.

En el caso de la Figura 5.13, se puede observar la interfaz tras presionar el botón de bajada. Gracias a la inhabilitación del resto de controles que no pertenecen a la secuencia de bajada, se garantiza que no se produzca ningún movimiento que la interrumpa. Este no es el único caso en el que se inhabilitan los controles, al presionar los demás botones se inhabilitan todos los controles de la página hasta que finalice la secuencia de movimientos. Una vez que el microcontrolador comunique la finalización de la acción, se vuelven a habilitar y se actualiza las posiciones de cada uno de los motores.

Clase index.html

La creación de la página web comienza definiendo su estructura a través del código HTML. Por ello, se crea el archivo *index.html*, el cual actúa como esqueleto definiendo los controles y sus características. En esta sección se definen los botones y barras deslizantes, definiendo la parte visual y funcional de cada uno. Este código, al ser el esqueleto de la aplicación hará uso del resto de clases de diseño y funcionalidad para completar la interfaz final.

Empezando por los botones, se crean dentro de un contenedor genérico definido por la clase *button-container*. Esta clase es utilizada por el resto de los códigos para facilitar su identificación y diseño. En este contenedor, se definen los cinco botones mediante la etiqueta *button*. Dicha etiqueta es usada para definir cada botón con un identificador, el texto que lo describe y la acción a ejecutar al ser pulsados.

```
<div class="button-container">
  <button id="resetButton" onclick="resetServo()">Reset</button>
</div>
<div class="button-container">
  <button id="lowerButton" onclick="lowerArm()">Bajar</button>
  <div id="lowerButtonOptions" class="hidden-button">
    <button id="gripButton" onclick="gripArm()">Agarrar</button>
    <button id="cancelMovementButton" onclick="cancelMovement()">Cancelar</button>
  </div>
  <button id="releaseButton" onclick="releaseArm()">Soltar</button>
</div>
```

Figura 5.14: Código HTML de los botones

En el caso de las barras deslizantes, al ser su función el desplazamiento de los seis motores de manera individual, se emplea un range-slider por cada uno. Estos también son almacenados dentro de la clase genérica *slider-container*. En cuanto a su configuración gráfica, se define tanto el texto como el icono en la parte superior de la barra; así como un valor entero que representa la posición actual de cada uno de los servomotores. Además, en el segmento *slider-labels*, se añaden dos etiquetas en los extremos de la barra que indican el efecto ocasionado en el actuador al desplazar el control hacia ese punto.

Para definir la lógica de estos controles se utiliza la etiqueta *input*, en la cual se establecen todas las funcionalidades de las barras deslizantes. En primer lugar, se define al componente como un objeto de tipo *range*, el cual lo establece como una barra deslizante, y se le atribuye un identificador a cada uno de los sliders. Por otro lado, se definen los valores mínimos y máximos a los que puede llegar el motor, limitando así el movimiento de cada uno. Gracias a esto, se obtienen movimientos más naturales y acotados, evitando desplazamientos que podrían desestabilizar la estructura del brazo.

Por último, se determinan los métodos a ejecutar al mover la barra, los cuales son accionados de dos maneras: mientras la barra es desplazada a través de *oninput* y al soltar el control con *onchange*. Dentro del primer evento, al desplazar el control, se ejecutan las funciones dedicadas a sincronizar la barra en el resto de las páginas web abiertas (*syncSlider*) y a accionar el motor aplicando un limitador de tráfico (*throttledMoveServo*). En cambio, *onchange* solo ejecuta la función para desplazar el servomotor sin el uso del limitador, mediante la función *moveServo*. Esto asegura que siempre se envíe el último ángulo pese a estar en un momento en el que el tráfico está restringido.

```
<div class="slider-container">
  <label
    >Servo Base
    <img src="images/baseImage.png" alt="baseIcon" class="label-icon" />
    <span id="baseValue" class="current-value">90°</span>
  </label>
  <div class="slider-labels">
    <span class="slider-left-label">Izquierda</span>
    <span class="slider-right-label">Derecha</span>
  </div>
  <input type="range" min="0" max="180" value="90" class="slider" id="baseSlider"
         oninput="syncSlider('base', this); throttledMoveServo('base', this)"
         onchange="moveServo('base', this)"
  />
</div>
```

Figura 5.15: Código HTML de las barras

Clase script.js

La clase *script.js* es la encargada de manipular la parte lógica de la página web desplegada. Este código maneja las funciones accionadas por los controles de la interfaz con el objetivo de enviar las peticiones correspondientes al servidor. Además, se encarga de manejar el intercambio de mensajes a través de WebSocket para actualizar la interfaz web. Por ello, este segmento toma un papel esencial, ya que actúa como intermediario entre la parte visual de la página y el back-end.

Empezando por los controles, los botones ejecutan distintas funciones al ser presionados. Estas funciones son *resetServo*, *lowerArm*, *gripArm*, *cancelMovement* y *releaseArm*; cada una asociada a un botón. El objetivo de cada una de ellas es comunicarle al microcontrolador, a través del back-end, la petición para realizar dicha secuencia de movimientos. Además, con el objetivo de deshabilitar los controles al presionar un botón, envía un mensaje al socket para actualizar la interfaz de todas las páginas abiertas.

Las peticiones para accionar la secuencia de movimientos se realizan mediante el protocolo HTTP, con la finalidad de enviar dicho mensaje al back-end. Mediante la librería axios se realiza una consulta GET. Esta no contiene ningún cuerpo de mensaje, sin embargo, su ruta indica al servidor el mensaje que debe realizar al broker MQTT para su posterior procesamiento en el microcontrolador.

Una vez que el back-end comunica que el mensaje ha sido publicado en el broker, se realiza el mensaje a través de WebSocket. En este se envía un objeto JSON, donde se indica el tipo de mensaje y el botón pulsado. Este mensaje es procesado por la parte del back-end, la cual responde con una notificación a todas las páginas abiertas para deshabilitar los controles.

La siguiente figura representa el código de la función ejecutada al presionar el botón “Reset”. El resto de las funciones siguen el mismo patrón, cambiando la ruta de la petición y el nombre del botón en la sentencia *wss.send*.

```
function resetServo() {
  axios
    .get(`/reset`)
    .then((response) => {
      console.log("Mensaje RESET enviado al microcontrolador");
      if (wss.readyState == wss.OPEN) {
        wss.send(JSON.stringify({ type: "buttonAction", action: "resetButton" }));
      }
    })
    .catch((error) => {
      console.error("Error al enviar mensaje RESET:", error);
    });
}
```

Figura 5.16: Función resetServo

En el caso de las barras deslizantes se ejecutan tres métodos al ser accionadas. El primero de ellos, *syncSlider*, se encarga de sincronizar la posición de una barra deslizante en todas las páginas web desplegadas. Esto permite que, si hay varios usuarios conectados a la aplicación web, todos puedan observar el desplazamiento que se realiza sobre el control desde otro dispositivo. Mediante este sistema es posible visualizar que el brazo robótico está siendo accionado.

La función *syncSlider* se encarga de recibir el identificador y el objeto de la barra deslizante. A través de estos parámetros se obtiene tanto el índice como el ángulo al que ha sido desplazado. Estos datos son almacenados en un documento JSON el cual será enviado a través de WebSocket. Antes de ser enviado se comprueba la conexión con el servidor a través de *wss.readyState* asegurando el envío del mensaje.

```
function syncSlider(id, slider) {
  const servoIndex = servoIndexMap[id];

  if (wss.readyState == wss.OPEN) {
    wss.send(
      JSON.stringify({
        type: "sliderUpdate",
        data: { servo: servoIndex, angle: slider.value },
      })
    );
  }
}
```

Figura 5.17: Función syncSlider

Aparte de la sincronización de la barra, la funcionalidad más importante es el accionamiento de un motor del brazo robótico. Por ello, el front-end se encarga de realizar peticiones HTTP a través de la función *moveServo*. Al ser ejecutada la función, recibe los parámetros del slider con el objetivo de obtener el identificador del control y el ángulo objetivo. Estos datos son enviados al servidor mediante una petición POST, la cual será recibida por el back-end y enviada al broker MQTT para su actuación en el microcontrolador.

Teniendo en cuenta que este método es ejecutado al desplazar la barra, la velocidad en la que sea accionada puede suponer un problema. Si se mueve la barra a una alta velocidad podría proporcionar una sobrecarga en la comunicación con el servidor. Por ello, se define un limitador de frecuencia (throttle), el cual restringe el envío de solicitudes a una cada 200 milisegundos. Este limitador es aplicado en la función comentada anteriormente dando como resultado *throttleMoveServo*. De este modo se garantiza un control eficiente sobre el envío de peticiones.

```
function throttle(func, delay) {
  let timer = 0;
  return function () {
    const now = Date.now();
    if (now - timer >= delay) {
      timer = now;
      func.apply(this, arguments);
    }
  };
}

function moveServo(id, slider) {
  document.getElementById(`${id}Value`).textContent = `${slider.value}°`;
  const servoIndex = servoIndexMap[id];
  axios
    .post("/moveServo", {
      servo: servoIndex,
      angle: slider.value,
    })
    .then((response) => {
      console.log(`Servo ${id} actualizado con éxito:`, response.data);
    })
    .catch((error) => {
      console.error(`Error al actualizar el servo ${id}:`, error);
    });
}

const throttledMoveServo = throttle((id, slider) => moveServo(id, slider), 200);
```

Figura 5.18: Petición HTTP para mover un servomotor

Finalizada la programación de los controles, la siguiente parte en el desarrollo del código es la gestión de los mensajes recibidos a través el WebSocket para actualizar la interfaz de la aplicación. Para ello, se establece la conexión al servidor mediante la dirección y puerto en el que se encuentra alojado. Posteriormente, se define el comportamiento frente a la recepción de los mensajes a través del método *wss.onmessage*. Este se activa al recibir un evento del servidor, permitiendo ejecutar diferentes acciones según el tipo de mensaje obtenido. Para procesarlo correctamente, el mensaje se convierte a un formato más adecuado, facilitando la identificación de su propósito para ejecutar la acción correspondiente.

A continuación, se desglosan cada una de las funciones utilizadas:

```
wss.onmessage = (event) => {
  const message = JSON.parse(event.data);
  if (message.type === "interfaceUpdate") {
    const lowerButtonState = message.data.lowerButtonState;
    const controlState = message.data.controlState;
    if (message.data.angles != undefined) {
      const angles = message.data.angles;
      updateSliders(angles);
    }
    updateButtons(lowerButtonState);
    if (controlState == "disabledNormal") {
      disableControls(true);
    } else if (controlState == "enabledNormal") {
      disableControls(false);
    } else {
      enableLowerControl();
    }
  } else if (message.type === "sliderUpdate") {
    const { servo, angle } = message.data;
    const servoName = Object.keys(servoIndexMap).find(
      (key) => servoIndexMap[key] === servo
    );
    if (servoName) {
      const slider = document.getElementById(`${servoName}Slider`);
      slider.value = angle;
      document.getElementById(`${servoName}Value`).textContent = `${angle}°`;
    }
  } else {
    console.error("Datos de ángulos inválidos recibidos:", message);
  }
};
```

Figura 5.19: Manejo WebSocket del front-end

- **updateSliders:** Se encarga de actualizar los valores de todas las barras mediante los ángulos recibidos desde el ESP8266. Por ello, se define un bucle en el cual se cambia el valor del ángulo y la posición de todas las barras. Solo es ejecutada si el mensaje del WebSocket contiene el parámetro de ángulos.

```
function updateSliders(angles) {
  Object.keys(servoIndexMap).forEach((servoName, index) => {
    const angle = angles[index];
    document.getElementById(`${servoName}Value`).textContent = `${angle}°`;
    document.getElementById(`${servoName}Slider`).value = angle;
  });
}
```

Figura 5.20: Actualización de las barras

- **updateButtons:** Esta función controla el estado de los botones de la secuencia de bajada. Si el atributo *lowerButtonState* recibido contiene la cadena de caracteres "hidden" desactiva el botón de bajada para habilitar los botones "Agarrar" y "Cancelar". De lo contrario, habilita el botón de bajada y oculta los otros dos.

```
function updateButtons(lowerButton) {
  if (lowerButton == "hidden") {
    document.getElementById("lowerButton").classList.add("hidden-button");
    document
      .getElementById("lowerButtonOptions")
      .classList.remove("hidden-button");
  } else {
    document
      .getElementById("lowerButtonOptions")
      .classList.add("hidden-button");
    document.getElementById("lowerButton").classList.remove("hidden-button");
  }
}
```

Figura 5.21: Actualización de los botones de bajada

- **disableControls:** Según el parámetro de entrada, habilita o deshabilita todos los controles. Esta función recibe como parámetro una variable de tipo booleano la cual indica la actualización a realizar. Mediante dos bucles, se actualiza el estado de los sliders y los botones.

```
function disableControls(disable) {
  const sliders = document.querySelectorAll(".slider");
  const buttons = document.querySelectorAll(".button-container button");
  sliders.forEach((slider) => {
    slider.disabled = disable;
  });
  buttons.forEach((button) => {
    button.disabled = disable;
  });
}
```

Figura 5.22: Estado de los controles

- **enableLowerControl:** Función ejecutada tras pulsar el botón de bajada, se encarga de deshabilitar todos los controles, menos los dos botones de agarrar un objeto o cancelar bajada. Ya que todas las barras tienen que permanecer inhabilitadas en esta fase, se crea un bucle que actualiza cada una de ellas. Posteriormente, se deshabilitan todos los botones mediante un bucle, exceptuando los de "Agarrar" y "Cancelar".

```
function enableLowerControl() {
  const sliders = document.querySelectorAll(".slider");
  const buttons = document.querySelectorAll(".button-container button");

  sliders.forEach((slider) => {
    slider.disabled = true;
  });

  buttons.forEach((button) => {
    if (button.id == "gripButton" || button.id == "cancelMovementButton") {
      button.disabled = false;
    } else {
      button.disabled = true;
    }
  });
}
```

Figura 5.23: Inhabilitación de controles tras bajada

La última parte de este código es la función que se ejecutará una vez la página haya sido completamente cargada. Al ser iniciada, es importante poder visualizar al instante el estado en el que se encuentra el brazo robótico. Por ello, se desarrolla la función *updateAllServos*, la cual permite actualizar la página al ser cargada mediante una petición al servidor web.

Esta función se encarga de enviar una petición GET la cual será procesada por el back-end para su comunicación al microcontrolador. El microcontrolador responde a esta petición enviando la posición de cada uno de los motores al servidor. Una vez obtenido los ángulos, el servidor envía esta información junto con el estado de los controles a la página web a través de WebSocket. El front-end es el encargado de recibir este evento y configurar cada una de las partes de su interfaz. Gracias a esto, el usuario final puede observar el estado del brazo robótico al acceder a la web.

```
function updateAllServos() {
  axios
    .get(`/servoAngles`)
    .then((response) => {
      console.log("Angulos de los servos recibidos:", response.data);
    })
    .catch((error) => {
      console.error("Error al recibir los ángulos de los servos:", error);
    });
}
window.onload = updateAllServos;
```

Figura 5.24: Petición ángulos de la web

5.2.2. Desarrollo del servidor web

El back-end se encarga de gestionar todas las consultas de la página web y actúa como intermediario con el microcontrolador. A parte de esto, se encarga de controlar la comunicación a través del socket para actualizar la interfaz gráfica de la web. Por otro lado, maneja la publicación de métricas del sistema para su almacenaje en la base de datos.

Para el desarrollo del servidor se han hecho uso de varias librerías:

- **Express:** Framework de NodeJS, se encarga crear el servidor, actuar como middleware y definir las rutas HTTP para recibir los mensajes de la página web.
- **Path:** Gestiona los archivos de la página web almacenados en el directorio public para su ejecución.
- **MQTT:** Controla todas las etapas de configuración del protocolo, desde la suscripción y recepción de mensajes, hasta su publicación.
- **SystemInformation:** Utilizado para obtener la información del sistema, como el uso de memoria y CPU por parte de la aplicación.
- **WebSocketServer:** Librería encargada de crear el servidor WebSocket, realizar la conexión y controlar el intercambio de mensajes con la web mediante esta tecnología.
- **File System:** Permite la lectura y escritura sobre el archivo *web_status.json* encargado de almacenar el estado de la interfaz web.

Clase index.js

Esta clase actúa como middleware para manejar las peticiones HTTP enviadas desde la página web. Con este propósito se hace uso del método *router* de la librería Express para establecer un flujo de ejecución cuando se recibe una solicitud. Para ello, se define una ruta por cada petición que puede enviar la capa de usuario. Dentro de estas solicitudes se pueden diferenciar dos tipos para la comunicación entre ambas partes de la plataforma.

Por un lado, se ha desarrollado un punto final para recibir la petición que permite desplazar un servomotor. Esta recibe una consulta POST en la cual contiene como cuerpo de la solicitud el índice del motor y el ángulo objetivo. Mediante la obtención de estos datos se genera un documento JSON el cual es publicado en el tema *esp8266/moveServo* del servidor MQTT.

```
router.post("/moveServo", async (req, res) => {
  const { servo, angle } = req.body;

  if (servo === undefined || angle === undefined) {
    return res.status(400).send("Faltan los parámetros necesarios");
  }
  const topic = `esp8266/moveServo`;
  const message = JSON.stringify({ servo, angle });

  mqttClient.publish(topic, message, (err) => {
    if (err) {
      console.error("Error al publicar MQTT", err);
      return res.status(500).send("Error al mover el servo");
    }
    console.log(`Servo ${servo} movido al ángulo ${angle}`);
    res.send("Servo movido con éxito");
  });
});
```

Figura 5.25: Ruta moveServo back-end

Por otro lado, para el resto de las peticiones se crea un punto final a través del protocolo HTTP GET. Estos se diferencian por no contener ningún tipo de información en la petición enviada. El flujo de estas funciones se establece identificando la solicitud expedida mediante la ruta y, conforme a ello, enviando el mensaje al broker para su recepción en el microcontrolador. En la siguiente figura se puede observar la función utilizada para la petición del botón "Reset". El resto de las funciones que utilizan este protocolo mantienen la misma estructura, cambiando el nombre de la ruta, tema y contenido del mensaje a enviar.

```
router.get("/reset", async (req, res) => {
  const topic = `esp8266/predefinedMovement/request`;
  const message = JSON.stringify({ movement: "resetBrazo" });

  mqttClient.publish(topic, message, (err) => {
    if (err) {
      console.log("Error al publicar MQTT", err);
      return res.status(500).send("Error al resetear el brazo");
    }
    console.log(`Comando para resetear el brazo enviado con éxito`);
    res.send("Brazo reseteado con éxito");
  });
});
```

Figura 5.26: Ruta reset del back-end

Clase mqtt.js

La clase mqtt.js es la encargada de manejar la configuración del servidor con la plataforma MQTT utilizada. Para ello, se ha desarrollado el método *setupMQTT*, el cual establece todos los parámetros necesarios para su configuración. En dicha función se realiza la conexión a la plataforma mediante el método *mqtt.connect*, donde se le indica la dirección y el puerto en el que está alojado.

Tras esto, se realiza la suscripción a los temas necesarios para el funcionamiento del servidor. En este caso, los temas a los que se suscribe son los encargados de enviar la posición del brazo robótico al iniciar el microcontrolador, finalizar un movimiento predefinido y al cargar la página web. Por otro lado, se inicializa la función *sendSystemMetrics*, la cual es la encargada de enviar las métricas de estado del servidor.

```
const mqtt_url = "mqtt://192.168.1.137:1883";
const topics = [
  "esp8266/predefinedMovement/response",
  "esp8266/servoAngles/response",
  "esp8266/startArm",
];

function setupMQTT() {
  const mqttClient = mqtt.connect(mqtt_url);
  mqttClient.on("connect", () => {
    topics.forEach((topic) => {
      mqttClient.subscribe(topic, (err) => {
        if (!err) {
          console.log(`Suscrito al topic ${topic}`);
        } else {
          console.error(`Error al suscribirse al topic ${topic}`, err);
        }
      });
    });
    sendSystemMetrics(mqttClient);
  });
  mqttClient.on("error", (err) => {
    console.error(`Error conectarse al broker`, err);
  });
  return mqttClient;
}
```

Figura 5.27: Inicialización MQTT del back-end

Para obtener los datos referentes al estado del sistema se utiliza la función *getSystemMetrics*. En este código se obtiene el porcentaje de uso del procesador a través de *si.currentLoad* y se calcula el uso de memoria convirtiéndolo en megabytes. A parte, obtiene el número de usuarios conectados a la página web a través de la variable global definida en la clase destinada para el WebSocket. Estos datos son retornados por la función para ser enviados al tema habilitado del broker a través de *sendSystemMetrics*.

```
async function getSystemMetrics() {
  try {
    const cpu = await si.currentLoad();
    const memoryUsage = process.memoryUsage();
    const memoryUsed = memoryUsage.rss;
    const memoryUsedMB = Math.round((memoryUsed / 1024 / 1024) * 100) / 100;
    const totalUsersWS = global.WS_CLIENTS ? global.WS_CLIENTS.size : 0;
    return {
      cpuUsage: cpu.currentLoad,
      memoryUsed: memoryUsedMB,
      totalUsers: totalUsersWS,
    };
  } catch (error) {
    console.error("Error al obtener métricas del sistema:", error);
    return null;
  }
}
```

Figura 5.28: Recolección métricas del sistema

Clase websocket.js

Con el objetivo de manejar el intercambio de mensajes para actualizar la interfaz de todos los dispositivos conectados, se ha creado la clase *websocket.js*. Esta clase se encarga de establecer la conexión y configuración previa, así como de la ejecución de los procesos a realizar frente a los posibles eventos.

En esta clase se hace uso el archivo *web_status*, el cual almacena el último estado visual de los controles. El objetivo de este documento es guardar dichos datos para ser enviados a la página web si se realiza una nueva conexión o se recarga la página. Gracias a esto, al acceder a la página web, permite observar el estado actual de cada uno de los controles. Para ello, se definen dos valores los cuales se encargan de almacenar la visualización de las barras y los botones. Los identificadores que cuenta el archivo json son los siguientes:

- **controlState:** Se encarga de gestionar el estado de habilitación o inhabilitación de los botones y las barras deslizantes. Pueden contener tres distintos valores:
  - **enabledNormal:** Variable que especifica que todos los controles son habilitados.
  - **enabledLower:** Indica que solo se habilitan los botones de "Agarrar" y "Cancelar" tras realizar la bajada.
  - **disabledNormal:** Especifica que todos los controles permanecen deshabilitados.
- **lowerButtonState:** Destinado para almacenar el estado de los tres botones de la secuencia de bajada. Esta etiqueta puede contener dos valores:
  - **hidden:** Indica que el botón "Bajar" permanece oculto, mientras que se pueden observar los botones "Agarrar" y "Cancelar".
  - **visible:** Mantiene el botón de bajada visible, haciendo que el resto estén ocultos.

Esta clase es inicializada mediante el método *setupWebSocket*, el cual recibe desde la clase principal (*app.js*) el servidor en el que está alojado el socket y el cliente de MQTT para recibir los mensajes de los temas suscritos. El flujo inicial de dicha función se encarga de la conexión y configuración con el socket. Al realizar la conexión, se dispone a manejar los mensajes recibidos desde el microcontrolador y la aplicación web para responder con la actualización de la interfaz web.

```
function setupWebSocket(server, mqttClient) {
  const wss = new WebSocketServer({ server });
  wss.on("connection", (ws) => {
    console.log("WebSocket conectado");
    ws.on("message", (message) => {
      try {
        const data = JSON.parse(message.toString());
        if (data.type === "buttonAction") {
          handleButtonAction(wss, data);
        }
        if (data.type === "sliderUpdate") {
          sendWebSocket(wss, data);
        }
      } catch (error) {
        console.log("Error al procesar mensaje WebSocket:", error);
      }
    });
    ws.on("close", () => {
      console.log("WebSocket desconectado");
    });
    ws.on("error", (error) => {
      console.error("Error en WebSocket:", error);
    });
  });
  global.WS_CLIENTS = wss.clients;
  handleMQTTMessages(mqttClient, wss);
}
```

Figura 5.29: Inicialización WebSocket del back-end

Los mensajes recibidos desde la página web se diferencian entre los ejecutados al presionar un botón o al deslizar una barra. En el caso de la barra, el front-end envía un mensaje para sincronizar su posición con el resto de las páginas web conectadas al servidor del WebSocket. En este mensaje se recibe el ángulo y el identificador del control, siendo ambos datos reenviados a todas las aplicaciones desplegadas en ese momento. Para ello, se hace uso de la función *sendWebSocket*, la cual facilita el envío de mensajes realizados desde el lado del back-end.

Al ser presionado un botón, se ejecuta la función *handleButtonAction* encargada de enviar la inhabilitación de todos los controles y el estado de visualización de los botones correspondientes a la secuencia de bajada. Esta función actúa según el botón pulsado, en el caso de haber sido pulsado el botón de bajada se oculta este botón y aparcaren visibles los botones de “Agarrar” y “Cancelar”. En el caso de pulsar el resto de los botones, se ocultan los botones para sujetar un objeto y cancelar la bajada, manteniendo visible el botón “Bajar”. Una vez definido el estado de todos los controles, se almacenan estos valores en el archivo JSON, a través de *updateStatusChache,* y son enviados a las páginas web desplegadas.

```
function handleButtonAction(wss, message) {
  const action = message.action;
  let status = {
    controlState: "disabledNormal",
    lowerButtonState: action !== "lowerButton" ? "visible" : "hidden",
  };

  sendWebSocket(wss, { type: "interfaceUpdate", data: status });
  updateStatusCache(web_status_path, status);
}
```

Figura 5.30: Estado de los botones back-end

Como se puede observar en la Figura 5.29, el método *setupWebSocket* también se encarga de manejar los mensajes recibidos desde el broker MQTT. Para ello, se desarrolla la función *handleMQTTMessages*, la cual intercepta cada uno de los mensajes publicados en los temas que se ha suscrito el servidor. En este método se define la actuación frente a los posibles eventos. Por este motivo, al recibir un mensaje se diferencia el tema del que proviene y se actúa en consecuencia para enviar la información necesaria a las interfaces web.

```
function handleMQTTMessages(mqttClient, wss) {
  mqttClient.on("message", (topic, message) => {
    try {
      const parsedMessage = JSON.parse(message.toString());
      if (topic == "esp8266/servoAngles/response") {
        handleServoAnglesResponse(wss, parsedMessage);
      } else if (topic == "esp8266/predefinedMovement/response") {
        handlePredefinedMovementResponse(wss, parsedMessage);
      } else if (topic == "esp8266/startArm") {
        handleStartArm(wss, parsedMessage);
      }
    } catch (error) {
      console.error("Error al procesar mensajes de MQTT", error);
    }
  });
}
```

Figura 5.31: Manejo de mensajes MQTT

A continuación se desglosan cada uno los mensajes y su flujo en cada caso:

- **Respuesta ServoAngles:** Respuesta a la petición para obtener la posición del brazo robótico que la página web realiza al ser cargada. El back-end envía los ángulos, junto con los valores leídos del archivo *web_status*.

```
function handleServoAnglesResponse(wss, parsedMessage) {
  fs.readFile(web_status_path, function (error, data) {
    if (error) {
      console.error("Error al leer web_status", error);
      return;
    }
    const status = JSON.parse(data);
    const response = {
      angles: parsedMessage.angles,
      controlState: status.controlState,
      lowerButtonState: status.lowerButtonState,
    };
    sendWebSocket(wss, { type: "interfaceUpdate", data: response });
  });
}
```

Figura 5.32: Respuesta petición de ángulos

- **Respuesta PredefinedMovement:** Envía los ángulos recibidos como respuesta al finalizar una secuencia activada por los botones. Dependiendo de la acción realizada, modifica el estado de los controles de la interfaz y lo envía junto con los ángulos recibidos, modificando el fichero *web_status*.

```
function handlePredefinedMovementResponse(wss, parsedMessage) {
  let updatedMessage = {
    ...parsedMessage,
    controlState:
      parsedMessage.movement !== "bajarBrazo" ? "enabledNormal" : "enabledLower",
    lowerButtonState:
      parsedMessage.movement !== "bajarBrazo" ? "visible" : "hidden",
  };
  sendWebSocket(wss, { type: "interfaceUpdate", data: updatedMessage });
  const status = {
    controlState: updatedMessage.controlState,
    lowerButtonState: updatedMessage.lowerButtonState,
  };
  updateStatusCache(web_status_path, status);
}
```

Figura 5.33: Respuesta movimientos predefinidos

- **Inicio del brazo:** Mensaje que indica el arranque del brazo robótico. Al recibir este mensaje se reinicia la página a su estado original, manteniendo habilitados los controles y el botón de bajada visible. Envía dichos datos junto con la posición del brazo y almacena los nuevos valores en el archivo JSON.

```
function handleStartArm(wss, parsedMessage) {
  const updatedMessage = {
    ...parsedMessage,
    controlState: "enabledNormal",
    lowerButtonState: "visible",
  };
  sendWebSocket(wss, { type: "interfaceUpdate", data: updatedMessage });
  const status = {
    controlState: updatedMessage.controlState,
    lowerButtonState: updatedMessage.lowerButtonState,
  };
  updateStatusCache(web_status_path, status);
}
```

Figura 5.34: Respuesta inicio del microcontrolador

Clase app.js

A través de la clase principal *app.js* se inicializan el resto de las clases mencionadas anteriormente y se configura el servidor web. Para ello, se monta el servidor en el puerto 4000, estableciendo los archivos de la página web en la dirección raíz. Esto permite el acceso a la aplicación introduciendo la dirección y puerto del servidor desde cualquier buscador.

```
app.use(express.static(path.join(__dirname, "public/web")));
app.use(express.json());
app.use(logger("dev"));
app.use("/", router(mqttClient));
server.listen(PORT, () => {
  console.log(`Servidor ejecutándose en el puerto ${PORT}`);
});
```

Figura 5.35: Configuración del servidor

5.3. Almacenamiento y visualización de datos

5.3.1. Desarrollo de la base de datos

Esta parte del proyecto va dedicada a la implementación de la base de datos en InfluxDB. Una base de datos de series temporales en la que se almacenan el histórico de ángulos del brazo, los movimientos realizados y las métricas de estado del servidor. Para trabajar con esta herramienta el primer paso es su configuración mediante Docker Compose. A través del archivo *docker-compose.yaml* y un archivo *.env* se inicializan todas las variables necesarias para el funcionamiento de la base de datos. A continuación, se detallan cada una de ellas:

```yaml
version: '3.9'
services:
  influxdb:
    image: influxdb:latest
    ports:
      - "8086:8086"
    volumes:
      - ./data/influxdb/data:/var/lib/influxdb2
      - ./data/influxdb/config:/etc/influxdb2
    container_name: influxdb
    environment:
      DOCKER_INFLUXDB_INIT_MODE: setup
      DOCKER_INFLUXDB_INIT_USERNAME: ${INFLUXDB_USERNAME}
      DOCKER_INFLUXDB_INIT_PASSWORD: ${INFLUXDB_PASSWORD}
      DOCKER_INFLUXDB_INIT_ORG: ${INFLUXDB_ORG}
      DOCKER_INFLUXDB_INIT_BUCKET: ${INFLUXDB_BUCKET}
      DOCKER_INFLUXDB_INIT_ADMIN_TOKEN: ${INFLUXDB_TOKEN}
    networks:
      - monitoring
volumes:
  influxdb_data:
  grafana_data:

networks:
  monitoring:
```

Figura 5.36: Configuración de InfluxDB

- **Puerto:** La base de datos se monta en el puerto 8086 de la máquina. A través de este puerto se puede visualizar la interfaz web que dispone esta herramienta para controlar toda la información.
- **Nombre de usuario y contraseña:** Define las credenciales de administrador que permite acceder a la plataforma web.
- **Organización:** Espacio de trabajo para un grupo de usuarios. Agrupa todos los datos, cuadros de mando y cubos a través de la organización.
- **Bucket:** El cubo es la unidad donde se almacenan cada uno de los datos de series temporales.

- **Token:** El API token es una serie alfanumérica que permite una interacción segura entre la base de datos y el programa creado.

Una vez establecidos los parámetros de la base de datos, se inicia el desarrollo del middleware que maneja el almacenamiento de la información. Para ello se desarrolla el código *influxdb_mqtt.py*, el cual actúa como cliente MQTT para recibir los mensajes publicados en el broker y almacenarlos en la base de datos.

En primer lugar, se realiza la configuración de la base de datos a través del archivo de variables de entorno y el método de inicialización de la librería Influxdb_client. En esta configuración se indica la dirección, token y organización escritas en el archivo de variables de entorno. Después, se realiza la conexión con el servidor MQTT y se suscribe a los temas establecidos para esta parte del sistema.

```
INFLUXDB_BUCKET = os.getenv('INFLUXDB_BUCKET')
client = InfluxDBClient(url=os.getenv('INFLUXDB_URL'),
                        token=os.getenv('INFLUXDB_TOKEN'),
                        org=os.getenv('INFLUXDB_ORG'))
write_api = client.write_api()

MQTT_BROKER_URL = "IP"
MQTT_PORT = 1883
MQTT_TOPICS = [("esp8266/angles", 2), ("esp8266/moveServo", 2), ("esp8266/predefinedMovement/request", 2), ("system/metrics", 2)]

mqttc = mqtt.Client(mqtt.CallbackAPIVersion.VERSION2)
```

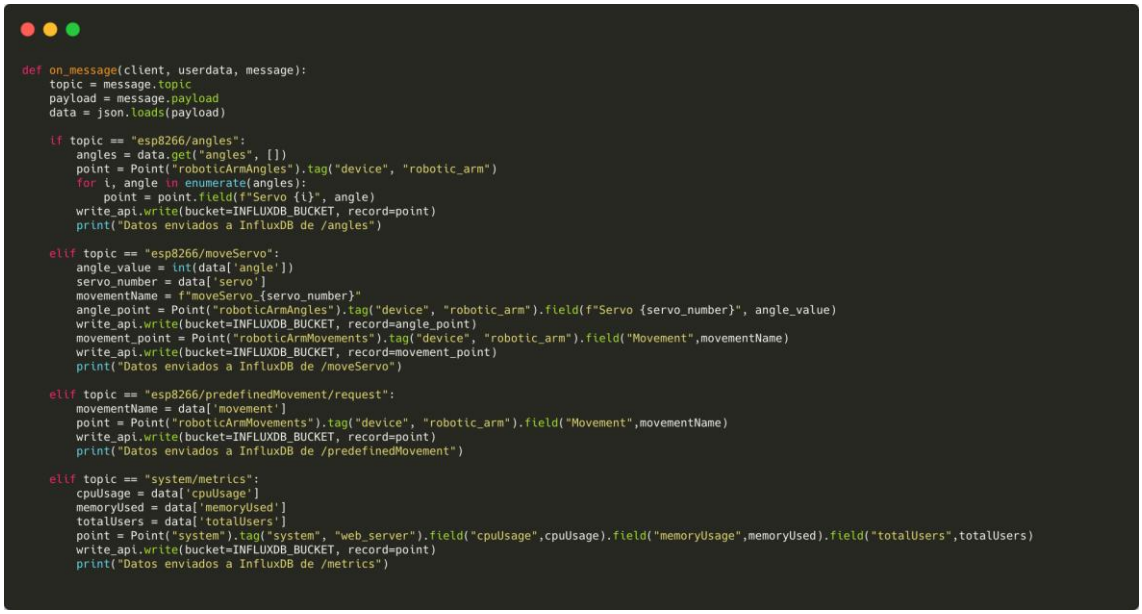
Figura 5.37: Configuración base de datos y broker

Al finalizar la configuración establecida para la base de datos y la plataforma de comunicación, se define el método necesario para el manejo de los mensajes recibidos. La función *on_message* de la librería utilizada para el control de MQTT se encarga de definir la actuación frente a los eventos escuchados en los temas suscritos. A través de este método se establece el flujo de ejecución según el tema del que se ha recibido el mensaje. Para ello, al interceptar el mensaje se realiza una conversión a un documento JSON, facilitando el proceso de obtención de la información recibida. Una vez generado el nuevo documento, se identifica el topic de origen mediante una estructura de control condicional para actuar en consecuencia.

A continuación, se describen cada uno de los temas suscritos y el posterior almacenamiento de la información recibida:

- **esp8266/angles:** Mensaje del microcontrolador, el cual envía la posición del brazo robótico. Almacena los datos recibidos en el registro *roboticArmAngles*, indicando el índice y ángulo correspondiente de cada servomotor.

- **esp8266/moveServo:** Tema en el cual se indica el desplazamiento de un motor. Se guarda el ángulo objetivo en la medición *roboticArmAngles*, así como el nombre del movimiento en *roboticArmMovements*.
- **esp8266/predefinedMovement:** Almacena el tipo de acción solicitada por la página web. Se almacena el nombre del movimiento, ya sea bajar, agarrar, cancelar bajada, soltar o reset, en la medición *roboticArmMovements*.
- **system/metrics:** Escucha el envío periódico de métricas del servidor web. Datos los cuales abarcan el número de usuarios conectados a la página, así como el uso de memoria y del procesador que realiza el servidor. Estos datos son almacenados en el registro *system* para su posterior visualización.



```
def on_message(client, userdata, message):
    topic = message.topic
    payload = message.payload
    data = json.loads(payload)

    if topic == "esp8266/angles":
        angles = data.get("angles", [])
        point = Point("roboticArmAngles").tag("device", "robotic_arm")
        for i, angle in enumerate(angles):
            point = point.field(f"Servo {i}", angle)
        write_api.write(bucket=INFLUXDB_BUCKET, record=point)
        print("Datos enviados a InfluxDB de /angles")

    elif topic == "esp8266/moveServo":
        angle_value = int(data['angle'])
        servo_number = data['servo']
        movementName = f"moveServo_{servo_number}"
        angle_point = Point("roboticArmAngles").tag("device", "robotic_arm").field(f"Servo {servo_number}", angle_value)
        write_api.write(bucket=INFLUXDB_BUCKET, record=angle_point)
        movement_point = Point("roboticArmMovements").tag("device", "robotic_arm").field("Movement",movementName)
        write_api.write(bucket=INFLUXDB_BUCKET, record=movement_point)
        print("Datos enviados a InfluxDB de /moveServo")

    elif topic == "esp8266/predefinedMovement/request":
        movementName = data['movement']
        point = Point("roboticArmMovements").tag("device", "robotic_arm").field("Movement",movementName)
        write_api.write(bucket=INFLUXDB_BUCKET, record=point)
        print("Datos enviados a InfluxDB de /predefinedMovement")

    elif topic == "system/metrics":
        cpuUsage = data['cpuUsage']
        memoryUsed = data['memoryUsed']
        totalUsers = data['totalUsers']
        point = Point("system").tag("system", "web_server").field("cpuUsage",cpuUsage).field("memoryUsage",memoryUsed).field("totalUsers",totalUsers)
        write_api.write(bucket=INFLUXDB_BUCKET, record=point)
        print("Datos enviados a InfluxDB de /metrics")
```

Figura 5.38: Recepción y almacenamiento en InfluxDB

Como se puede observar en la figura anterior, el almacenamiento a la base de datos se realiza mediante la función *Point*. En esta se indica el nombre del registro (measurement), la etiqueta (tag) y el valor a introducir (field). Mediante esta función se almacenan cada uno de los datos en respectivas colecciones para permanecer estructurados según su tipo.

La parte final del código se encarga de ejecutar el flujo de inicio y de bucle principal del programa. En este segmento se definen las funciones a ejecutar frente a los eventos de la comunicación MQTT. La función inicial realiza la conexión previa con la plataforma de comunicación al arrancar el programa. Una vez establecida la conexión se ejecuta continuamente el bucle principal, el cual se encarga de manejar la comunicación y la actuación frente a los mensajes recibidos.

5.3.2. Implementación de Grafana

Al implementar una base de datos para este proyecto, es recomendable utilizar una herramienta capaz de monitorear en tiempo real toda la información almacenada. Por ello, se ha elegido Grafana para la capa de visualización de la base de datos. A través de esta plataforma se genera un panel en el cual se implementan gráficas y tablas que permiten observar cada uno de los datos relevantes del sistema de manera sencilla e intuitiva.

Para que esta herramienta funcione correctamente, se ha establecido la configuración a través del mismo contenedor Docker Compose que el utilizado para la base de datos. En el código *docker-compose.yaml* se definen cada una de las variables necesarias para su funcionamiento y se establece una dependencia con la base de datos, la cual no permitirá la ejecución de Grafana si la base de datos no se encuentra operativa, evitando así posibles errores. En este trozo de código se establece el puerto 3000 para el montaje de la herramienta y se definen las credenciales de administrador. Esta configuración permite el acceso a la aplicación web de la herramienta, donde se puede visualizar el panel desplegado.

```
version: '3.9'
services:
  grafana:
    image: grafana/grafana:latest
    ports:
      - "3000:3000"
    volumes:
      - ./data/grafana:/var/lib/grafana
    container_name: grafana
    environment:
      - GF_SECURITY_ADMIN_USER=${GRAFANA_ADMIN_USER}
      - GF_SECURITY_ADMIN_PASSWORD=${GRAFANA_ADMIN_PASSWORD}
    networks:
      - monitoring
    depends_on:
      - influxdb
volumes:
  grafana_data:
networks:
  monitoring:
```

Figura 5.39: Configuración Grafana

Con el objetivo de que los datos almacenados sean visibles en el panel, se realiza la conexión de la base de datos con Grafana. Esta conexión se ha realizado siguiendo las instrucciones suministradas por los propios desarrolladores de la herramienta [28]. En esta configuración, se crea una fuente de datos detallando la herramienta utilizada para la base de datos, así como el puerto, organización, token y lenguaje de consulta. Una vez comprobado el correcto enlace entre ambas herramientas, se realiza la estructura del panel. En este panel se utilizan varios tipos de visualizadores para diferenciar las tres mediciones creadas en la base de datos.

A continuación, se especifican los visualizadores utilizados para cada uno de los datos almacenados:

- **Ángulos actuales:** Consta de seis paneles de estado que indican la última posición almacenada de cada uno de los motores del brazo.
- **Histórico de ángulos:** Se hace uso de una serie temporal que dibuja gráficamente el historial de posiciones de cada articulación del brazo a lo largo del tiempo.
- **Tabla de movimientos:** Una tabla que indica los veinte últimos movimientos solicitados a través de la página web.
- **Indicadores de rendimiento del servidor:** Mediante un panel de estado y dos gauges se indica el estado actual del servidor, correspondiente a la medición *system* creada para almacenar estos datos.

Para poder representar los datos en cada uno de visualizadores, Grafana ofrece un sistema de consultas con soporte para varios tipos de lenguaje. Debido a que este sistema despliega una base de datos a través de InfluxDB, se requiere del uso del lenguaje de consultas Flux. Estas consultas son realizadas mediante filtros en función de las etiquetas establecidas para la colección de datos. En la siguiente imagen se muestra la consulta usada para la gráfica de series temporales.

```
from(bucket: "TFG")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
  |> filter(fn: (r) => r["_measurement"] == "roboticArmAngles")
  |> filter(fn: (r) => r["device"] == "robotic_arm")
  |> fill(column: "_value", usePrevious: true)
  |> yield(name: "data")
```

Figura 5.40: Consulta del historial de ángulos

En esta consulta se definen los elementos a visualizar mediante el filtro de diferentes parámetros. El primer paso es escoger el cubo establecido para el alojamiento de todas las colecciones de datos, que en este caso se trata de "TFG". Posteriormente se define el rango de tiempo de los datos que se quieren visualizar, definiendo la obtención de todos los datos almacenados. Finalmente, se filtra la medición y el dispositivo en el que se encuentran los datos, que para este caso son los ángulos de los seis motores mediante la medición de *roboticArmAngles*. El resto de los componentes utilizados obtienen la información mediante la misma estructura, diferenciándose al cambiar los parámetros establecidos para el resto de las mediciones y tipos de datos.

6. Resultados

En esta sección se presentan las diferentes pruebas realizadas al sistema, evidenciando así su correcto funcionamiento. A través de estas pruebas y su posterior discusión se puede observar el cumplimiento de cada uno de los objetivos planteados al iniciar el proyecto.

6.1. Pruebas

A lo largo de toda la duración del proyecto se han ido realizando una serie de pruebas en las cuales se podía observar el funcionamiento del sistema frente a diferentes eventos. A través de dichas pruebas se han ido mejorando las distintas fases del sistema obteniendo un resultado final eficaz e intuitivo. Este apartado se encarga de enumerar cada una de las pruebas realizadas, así como su desarrollo y resultado final.

Teniendo en cuenta que el sistema solo es funcional si cada uno de los dispositivos están conectados a la misma red WiFi, se establecen todos ellos en una misma red local. Para estas pruebas se hace uso de un dispositivo móvil con el objetivo de acceder tanto a la aplicación web, para actuar y visualizar el estado del brazo robótico, como al panel de Grafana. Para observar las capturas de paquetes realizadas es esencial comprender las diferentes direcciones IP del sistema.

- **192.168.1.137:** Máquina en la que se ejecuta todo el sistema exceptuando la parte hardware. En esta se aloja tanto el servidor web, como el broker MQTT y la base de datos. Tanto el microcontrolador como el dispositivo móvil mantendrán un intercambio de mensajes con esta máquina para completar cada una de las solicitudes.
- **192.168.1.46:** Dispositivo móvil encargado de acceder a la aplicación web desplegada. A través de este dispositivo se puede realizar los movimientos sobre el brazo robótico a la vez que se visualiza su estado. Además, al estar en la misma red local, puede acceder a la capa de visualización establecida para la base de datos.
- **192.168.1.131:** Dirección del microcontrolador ESP8266 el cual actúa en estas pruebas como intermediario entre el servidor y el brazo robótico. El microprocesador es el encargado de recibir y publicar mensajes a través del broker MQTT, así como de la realización de los movimientos expedidos desde la aplicación.

- **192.168.4.1:** Dirección del microcontrolador al iniciar el punto de acceso una vez que no es capaz de realizar la conexión con la red inalámbrica. A través de este punto de acceso permite el ingreso a la página web dedicada para emitir las credenciales de la red.
- **192.168.4.2:** Dirección establecida para el dispositivo móvil al conectarse al punto de acceso. A través de este dispositivo se puede acceder a la página web de configuración de la red WiFi

En la siguiente tabla se especifican cada una de las pruebas realizadas con una breve descripción de cada una:

Prueba	Descripción
Configuración WiFi en ESP8266	El microprocesador debe conectarse a la red inalámbrica establecida con el propósito de mantener la comunicación con las distintas partes del sistema.
Comunicación MQTT entre componentes	Los tres segmentos del sistema deben permanecer conectados al servidor MQTT. Posteriormente, se realiza la suscripción a los temas destinados para la comunicación.
Desplazamiento de los motores	La página web solicita el desplazamiento de los servomotores individualmente y el microcontrolador los ejecuta.
Movimientos predefinidos	El brazo robótico tiene que estar capacitado para realizar la secuencia de movimientos solicitada por la web.
Visualización de la posición del brazo	El microcontrolador publica los ángulos del brazo robótico en el servidor MQTT. Dichos datos son emitidos a todas las páginas abiertas a través de WebSocket para su visualización.
Almacenamiento en la base de datos	El sistema creado para el almacenamiento de las métricas del sistema debe almacenar dicha información en la base de datos de InfluxDB.
Visualización de los datos en Grafana	Todos los datos almacenados son visualizados correctamente a través del panel creado en Grafana.

Tabla 6.1: Pruebas al sistema

1. Configuración WiFi en ESP8266

Una vez es iniciado el microcontrolador intenta realizar la conexión a la red establecida para el sistema. Para ello, lee las credenciales escritas en el archivo almacenado en su memoria interna. Si no se encuentra ninguna clave de acceso o la conexión es fallida, habilita un punto de acceso en el que se aloja un servidor web. Este servidor mantiene una interfaz RESTFul con el microcontrolador, la cual permite la colección de las credenciales suministradas desde la página web.

Esta prueba se realiza teniendo en cuenta que el archivo utilizado para almacenar las credenciales se encuentra vacío. Una vez es iniciado el sistema, no detecta ninguna credencial almacenada, con lo que dispone el punto de acceso. Desde cualquier dispositivo cercano se puede realizar la conexión al punto de acceso llamado "ConfigWiFiAP". Al realizar el emparejamiento con dicha red se posibilita el acceso a la página de configuración introduciendo la dirección IP del servidor que aloja microcontrolador.

Esta interfaz web está formada por un menú desplegable en el cual se muestran las redes detectadas por el microcontrolador. A través de este se elige la red deseada y se introduce su contraseña en la caja de texto inferior. Una vez escritas las credenciales se debe pulsar el botón "Guardar", el cual se encarga de enviar las credenciales escogidas al microcontrolador. Al presionar este botón se redirige a la ruta "/config", la cual confirma el envío de las credenciales.

Figura 6.1: Página configuración WiFi

A nivel de red, los datos son enviados a través de una solicitud POST en la ruta "/config", en la cual se recoge y se almacena la clave de acceso en la memoria interna del ESP8266. Una vez guardado, se reinicia el sistema para volver a intentar la conexión con la red expedida. Al ser la conexión exitosa, el microcontrolador inicia el protocolo de conexión con el servidor MQTT. Mediante el protocolo TCP con las banderas SYN y ACK, se asegura que la conexión ha sido establecida antes de realizar la suscripción a los temas. Cuando finaliza dicha confirmación, el microcontrolador se suscribe a los temas establecidos y publica el mensaje para comunicar su inicio. En la siguiente figura se pueden observar los paquetes capturados durante la prueba a través de la herramienta WireShark.

Source	Destination	Protocol	Info
192.168.4.2	192.168.4.1	HTTP	GET / HTTP/1.1
192.168.4.1	192.168.4.2	HTTP	HTTP/1.1 200 OK (text/html)
192.168.4.2	192.168.4.1	HTTP	POST /config HTTP/1.1 (application/x-www-form-urlencoded)
192.168.4.1	192.168.4.2	HTTP	HTTP/1.1 200 OK (text/html)
192.168.1.131	192.168.1.137	TCP	53618 → 1883 [SYN] Seq=0 Win=2144 Len=0 MSS=536 SACK_PERM
192.168.1.137	192.168.1.131	TCP	1883 → 53618 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 SACK_PERM
192.168.1.131	192.168.1.137	TCP	53618 → 1883 [ACK] Seq=1 Ack=1 Win=2144 Len=0
192.168.1.131	192.168.1.137	MQTT	Connect Command
192.168.1.137	192.168.1.131	TCP	1883 → 53618 [ACK] Seq=1 Ack=22 Win=65535 Len=0
192.168.1.137	192.168.1.131	MQTT	Connect Ack
192.168.1.131	192.168.1.137	MQTT	Subscribe Request (id=2) [esp8266/moveServo]
192.168.1.137	192.168.1.131	TCP	1883 → 53618 [ACK] Seq=5 Ack=46 Win=65535 Len=0
192.168.1.137	192.168.1.131	MQTT	Subscribe Ack (id=2)
192.168.1.131	192.168.1.137	MQTT	Subscribe Request (id=3) [esp8266/predefinedMovement/request], Subscribe Request (id=4) [esp8266/servoAngles/request]
192.168.1.137	192.168.1.131	TCP	1883 → 53618 [ACK] Seq=10 Ack=121 Win=65535 Len=0
192.168.1.131	192.168.1.137	MQTT	Publish Message [esp8266/startArm]
192.168.1.137	192.168.1.131	MQTT	Subscribe Ack (id=3), Subscribe Ack (id=4)

Figura 6.2: Paquetes de red al configurar la red

2. Comunicación MQTT entre componentes

Los diferentes componentes del sistema se comunican entre sí mediante el servidor MQTT desplegado. Por ello, es esencial asegurar la conexión de las tres partes con la plataforma de intercambio de mensajes. Gracias a esta, se presenta un sistema de comunicación sin latencia y persistente frente a diversos tipos de eventos durante la conexión.

Al ser iniciados, el control del brazo robótico, el servidor web y el script de la base de datos establecen conexión con el broker. Para realizar dicha conexión, cada una de las partes envían un mensaje de conexión al servidor MQTT, el cual la confirma a través del comando ACK. Tras la confirmación, cada una de las partes del sistema procede a la suscripción de los temas fijados.

Source	Destination	Protocol	Info
192.168.1.131	192.168.1.137	MQTT	Connect Command
192.168.1.137	192.168.1.131	MQTT	Connect Ack
192.168.1.131	192.168.1.137	MQTT	Subscribe Request (id=2) [esp8266/moveServo]
192.168.1.137	192.168.1.131	MQTT	Subscribe Ack (id=2)
192.168.1.131	192.168.1.137	MQTT	Subscribe Request (id=3) [esp8266/predefinedMovement/request], Subscribe Request (id=4) [esp8266/servoAngles/request]
192.168.1.131	192.168.1.137	MQTT	Publish Message [esp8266/startArm]
192.168.1.131	192.168.1.137	MQTT	Subscribe Ack (id=3), Subscribe Ack (id=4)
192.168.1.137	192.168.1.137	MQTT	Connect Command
192.168.1.137	192.168.1.137	MQTT	Connect Ack
192.168.1.137	192.168.1.137	MQTT	Subscribe Request (id=62404) [esp8266/predefinedMovement/response], Subscribe Request (id=62405) [esp8266/servoAngles/response], Subscribe Requ
192.168.1.137	192.168.1.137	MQTT	Subscribe Ack (id=62404)
192.168.1.137	192.168.1.137	MQTT	Subscribe Ack (id=62405)
192.168.1.137	192.168.1.137	MQTT	Subscribe Ack (id=62406)
192.168.1.137	192.168.1.137	MQTT	Connect Command
192.168.1.137	192.168.1.137	MQTT	Connect Ack
192.168.1.137	192.168.1.137	MQTT	Subscribe Request (id=1) [esp8266/angles] [esp8266/moveServo] [esp8266/predefinedMovement/request] [system/metrics]
192.168.1.137	192.168.1.137	MQTT	Subscribe Ack (id=1)

Figura 6.3: Conexión al iniciar el sistema

3. Desplazamiento de los motores

La página web tiene el control sobre los movimientos realizados por el brazo robótico. Para desplazar cada uno de los motores se emplean las barras deslizantes. Estas permiten accionar los actuadores individualmente a medida que se desplaza el control. Dicha actuación en tiempo real permite una mejor interacción entre ambas partes del sistema, permitiendo desplazar el motor a medida que se mueve el control de la barra.

El funcionamiento de dicha petición se divide en dos trazas, la petición HTTP y el mensaje enviado por WebSocket.

- **HTTP:** La página web emite una solicitud POST al servidor con el índice del motor y el ángulo objetivo. Estos datos son enviados al microprocesador a través del tema *esp8266/moveServo* para su actuación sobre el brazo robótico. Por último, el microprocesador escucha el mensaje recibido por el tema al que ha sido previamente suscrito y realiza un movimiento progresivo sobre dicho motor. El uso del limitador de frecuencia al desplazar la barra a altas velocidades resulta efectivo. Con la delimitación de envío de peticiones cada 200 milisegundos, se asegura que el estado de la comunicación sea estable, manteniendo un movimiento efectivo sin mostrar ninguna latencia entre el desplazamiento del control y el del motor.
- **WebSocket:** Este protocolo tiene como fin la sincronización de la barra deslizada en cada una de las páginas web abiertas. Para ello, se disponen varias páginas web y se procede a desplazar una de las barras en cualquier dispositivo. Dicho movimiento se ve reflejado a tiempo real en el resto de las páginas abiertas. Esto es posible mediante el mensaje que envía la página web que acciona el control. Este mensaje es procesado por el back-end para ser posteriormente enviado al resto de páginas abiertas. En dicho mensaje se envía la información necesaria a través de un objeto JSON, en el cual se indica la barra y posición desplazada mediante la etiqueta *servo* y *angle*.

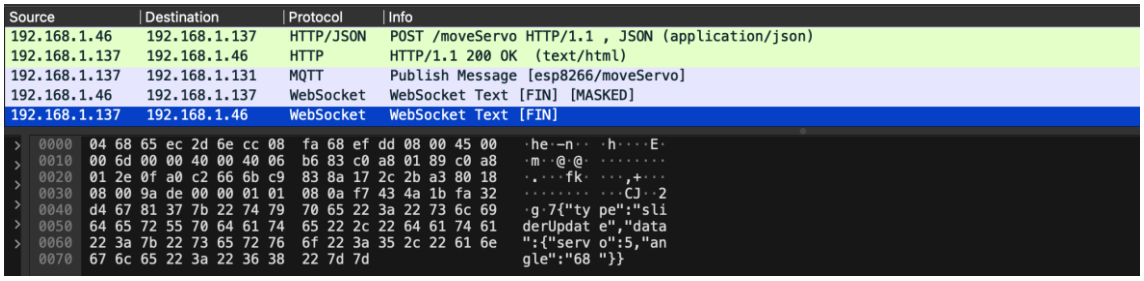


Figura 6.4: Paquetes de red al desplazar un motor

4. Movimientos predefinidos

El propósito de esta prueba es observar el correcto funcionamiento del brazo robótico al pulsar alguno de los botones. Al ser presionados se deben enviar dos mensajes diferentes. La petición para realizar la secuencia de movimientos a través de HTTP y la actualización de la interfaz en todos los dispositivos mediante WebSocket. El mensaje enviado a través del protocolo de hipertexto será recibido por el back-end para su publicación en la plataforma MQTT. El microcontrolador intercepta dicho mensaje y lleva a cabo la acción pedida.

A la hora de realizar esta prueba se ha decidido ejecutar la secuencia de bajada para comprobar todos los eventos posibles. Una vez pulsado el botón, la página web realiza la petición GET al back-end. Este la recibe y publica con éxito el mensaje en el broker para que el brazo realice la bajada.

Por otro lado, la petición a través de WebSocket indica el accionamiento del botón para que el resto de las páginas web actualicen su interfaz. Como se puede observar en la siguiente imagen, este mensaje contiene el archivo JSON que define la inhabilitación de todos los controles (*disablednormal*) y oculta el botón de bajada permitiendo observar los botones que continúan la secuencia. Este mensaje es recibido por todas las páginas web y actualizan su interfaz.

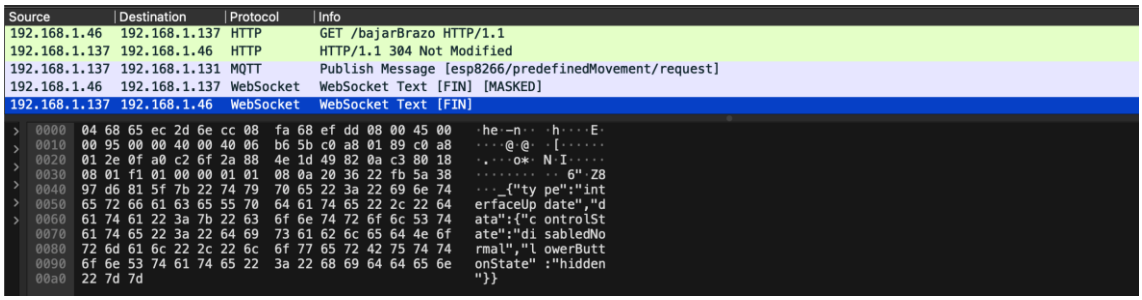


Figura 6.5: Paquetes de un movimiento predefinido

Mientras que la página mantiene todos los controles deshabilitados, el controlador recibe el mensaje publicado por el servidor web. Al interceptarlo identifica el tipo de acción a realizar y comienza la secuencia de movimientos. A través de señales PWM, abre la pinza del actuador final con el objetivo de agarrar un objeto. Una vez abierta completamente, desplaza los dos motores centrales del brazo robótico permitiendo la inclinación. Ambos son rotados hasta el ángulo 0, el cual es su ángulo mínimo y el que representa una bajada completa del manipulador. Cuando ha finalizado esta secuencia, el dispositivo responde a través del servidor MQTT enviando los ángulos actuales tanto al servidor como a la base de datos.

Estos ángulos son recibidos por el back-end, el cual los procesa y los envía a la interfaz web a través del servidor WebSocket desplegado. En este mensaje se especifican los ángulos recibidos y las variables de estado de los controles diseñadas para este caso. Dentro de estas variables destaca *enabledLower* la cual indica la inhabilitación de todos los controles exceptuando los botones "Agarrar" y "Cancelar". Y, por otro lado, se define la variable *hidden* la cual especifica que el botón de bajada debe permanecer oculto para mostrar los botones mencionados.

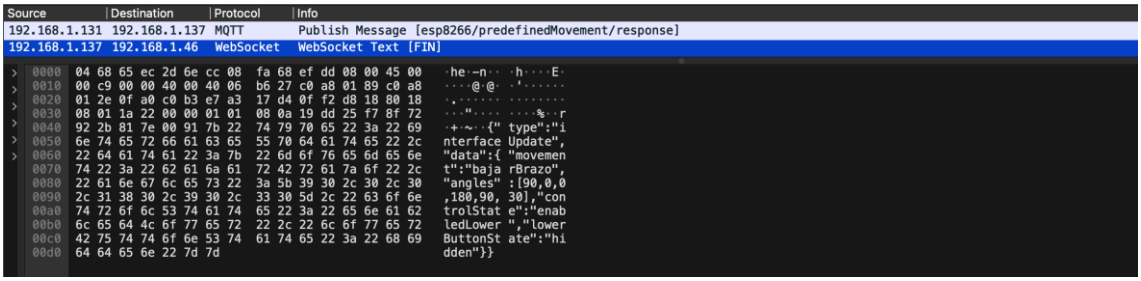


Figura 6.6: Respuesta al movimiento predefinido

Figura 6.7: Página al bajar el brazo

Como se puede observar en la Figura 6.7, la página actualiza correctamente su interfaz al recibir el mensaje. En este momento solo es posible finalizar la secuencia de bajada a través de la recogida del objeto o cancelando la bajada. Una vez accionada alguna una de las dos y finalizada la secuencia de movimiento, la página habilitaría todos los controles con la nueva posición del brazo. El resto de los movimientos predefinidos actúan de la misma manera, diferenciándose por el tipo de movimiento que desarrollan.

5. Visualización de la posición del brazo

Una funcionalidad de la página web es la posibilidad de observar la posición del brazo a través de ella. Tanto en los desplazamientos individuales accionados por las barras como al finalizar un movimiento predefinido, se actualizan todas las páginas abiertas con la posición del brazo. A parte, los controles toman un papel crucial para indicar el estado del manipulador. La inhabilitación de todos los controles indica que el brazo robótico está realizando una secuencia de movimiento. Además, como se ha podido observar en la prueba anterior, el sistema permite la gestión de estados visibles y no visibles de sus botones dependiendo de la secuencia.

Teniendo en cuenta un espacio colaborativo o una posible pérdida de conexión, es importante que al acceder de nuevo a la página se pueda observar el estado actual. Gracias a gestión del archivo *web_status* en el back-end, cada uno de estos aspectos de la interfaz son almacenados para su envío al iniciar la página. En esta prueba se observa el comportamiento al acceder de nuevo una vez se ha finalizado la secuencia de bajada, teniendo que obtener la misma interfaz que en la Figura 6.7.

Al ser iniciada la página web se realizan las peticiones GET para obtener los recursos adicionales de diseño (*style.css*), de funcionalidad (*script.js*) y las imágenes de cada uno de los motores almacenadas en el servidor. Al recibir estos recursos, se realiza la conexión con el servidor WebSocket. Una vez finalizada la configuración inicial, se realiza la petición para obtener la posición actual del brazo (*servoAngles*). Al recibir los ángulos, el back-end los envía al socket junto con el estado de los controles. Finalmente, la página es cargada correctamente y se observan todos los controles deshabilitados menos los de la secuencia de bajada.

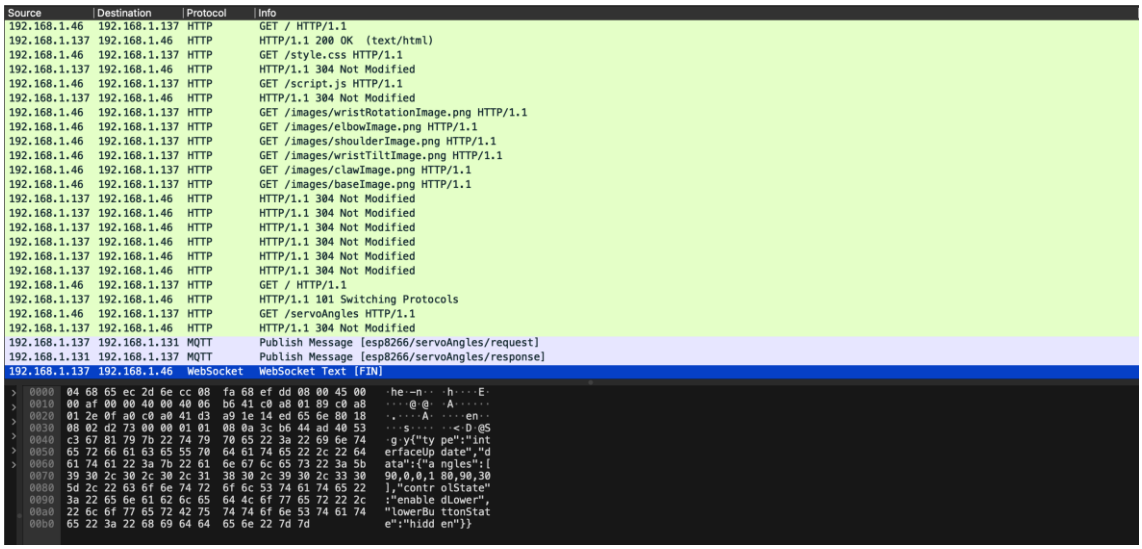


Figura 6.8: Paquetes al iniciar la página

6. Almacenamiento en la base de datos

Esta prueba se centra en comprobar si los datos recibidos desde MQTT son interpretados por el programa desarrollado en Python y son enviados correctamente a la base de datos. Este sistema está capacitado para escuchar cada uno de los mensajes publicados en los temas suscritos. Al escucharlos deben escribirlos a InfluxDB a través de la API que proporciona el desarrollador de esta herramienta. Esta sección se divide en cuatro pruebas correspondiente a cada uno de los temas suscritos:

1. **Ángulos del brazo:** La posición del brazo robótico es enviada de manera periódica y al finalizar un movimiento predefinido. En este mensaje se encuentra cada uno de los ángulos con su respectivo índice para identificar cada motor. El sistema desplegado escucha correctamente el mensaje y escribe dicha información en la tabla *roboticArmAngles*.

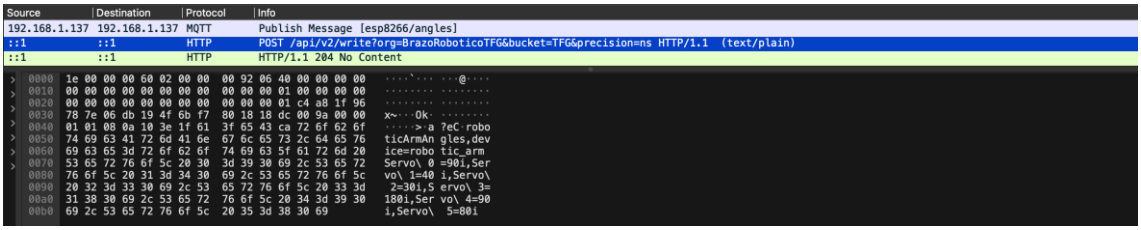


Figura 6.9: Almacenamiento ángulos

2. **Desplazamiento del actuador:** A la hora de desplazar un servomotor, se intercepta el mensaje publicado desde el servidor web. La escritura de estos datos se divide en dos campos, la posición del motor desplazado y el tipo de movimiento realizado. El primero es almacenado con éxito en *roboticArmAngles* indicando su número de índice y el ángulo. El segundo se guarda en *roboticArmMovement* indicando el tipo de movimiento realizado. En el caso de esta prueba se desplaza la barra número cuatro, correspondiente a rotación de la muñeca, resultando el mensaje en *moveServo_4*.

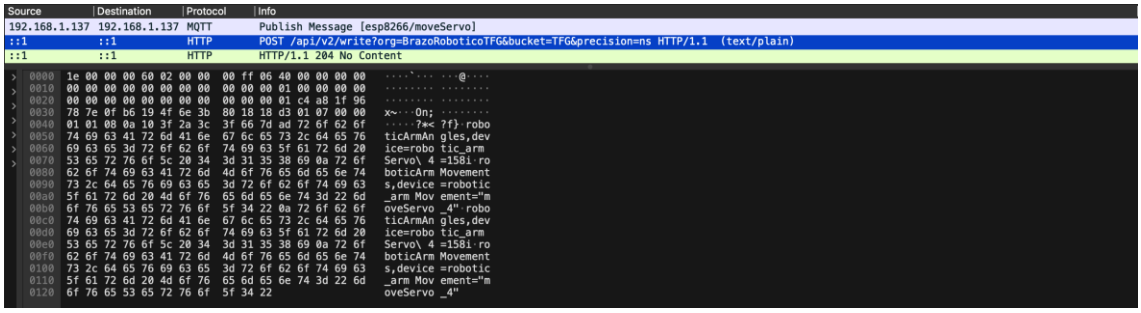


Figura 6.10: Almacenamiento del movimiento del actuador

3. **Movimientos predefinidos:** Al presionar un botón en la interfaz web, el back-end publica dicha petición en el broker MQTT. El programa creado en Python para la base de datos lo intercepta y lo almacena con éxito. Al igual que en el apartado anterior, el movimiento es guardado en *roboticArmMovement*. Para esta prueba se ha presionado el botón "Soltar" dando como resultado para su almacenaje la cadena de caracteres *soltarBrazo*.

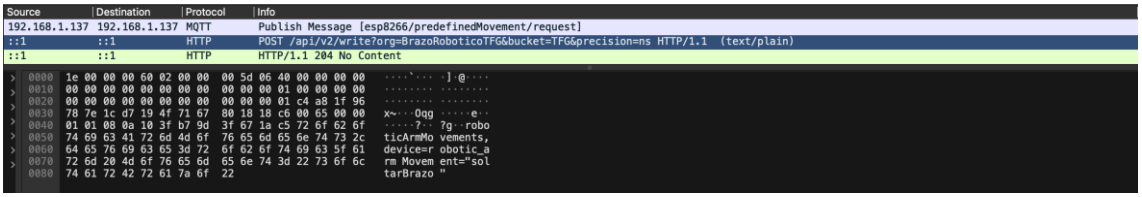


Figura 6.11: Almacenamiento movimientos predefinidos

4. **Métricas del servidor web:** El servidor envía cada una de las métricas de su sistema para su almacenaje. En este caso, se almacena toda la información en la tabla *system*. En esta tabla se guarda el porcentaje de uso del procesador (*cpuUsage*), la memoria usada (*memoryUsage*) y el número de usuarios conectados a la página web (*totalUsers*).

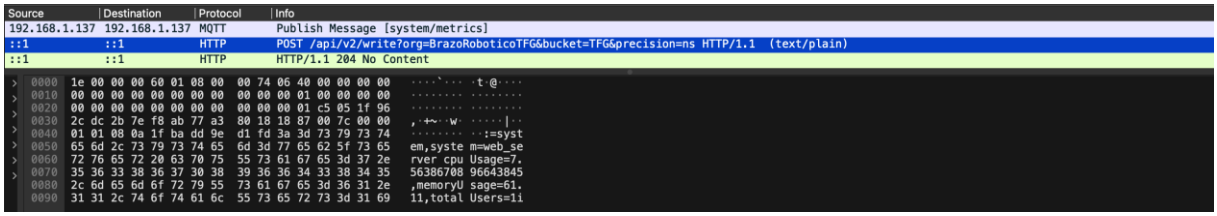


Figura 6.12: Almacenamiento métricas del servidor

7. Visualización de los datos en Grafana

Una vez que cada uno de los datos han sido correctamente almacenados, se debe probar su representación gráfica en Grafana. A través de un panel se han diseñado diferentes métricas para comprobar los datos claves del sistema. En este panel se pueden observar cada uno de los datos almacenados de la prueba anterior.

En el caso de los ángulos del brazo, se hace uso de un indicador por cada uno de los motores, representando la última posición almacenada. También se hace uso de una gráfica de series temporales que almacena el historial de la posición de cada uno de los motores. Ambos tipos de datos son extraídos con éxito de la tabla *roboticArmAngles* al realizar las consultas pertinentes a través del lenguaje Flux. Al realizar la prueba de la acción para soltar un objeto, se puede observar el almacenamiento de los ángulos al desplazar el brazo y al finalizar dicha secuencia.

Por otro lado, para visualizar los movimientos expedidos desde la página web se crea una tabla. Dicha tabla extrae con éxito el historial de movimientos expedidos con la fecha de ejecución. A través de esta tabla podemos observar las pruebas realizadas para almacenar el movimiento de un actuador a través de la cadena de caracteres *moveServo_4*. Así como, las peticiones para realizar las secuencias de movimiento como *resetBrazo* y *soltarBrazo*.

En último lugar, destacan las métricas que envía el servidor web en un intervalo de treinta milisegundos. Para su visualización se crea un indicador el cual muestra el número de usuarios conectados, que en el caso de la prueba realizada anteriormente es de uno. Por otra parte, se crean dos medidores los cuales representan los datos de porcentaje de uso de procesador y el uso de memoria en megabytes.

A través de este panel se puede visualizar cada uno de los datos almacenados en la base de datos con gran facilidad. La herramienta Grafana permite la monitorización de cada una de las partes del sistema a tiempo real. Esto se debe a que la herramienta es capaz de actualizar la información del panel a través de consultas periódicas. Gracias a esto, permite generar un seguimiento detallado del rendimiento de sistema de manera continuada y a simple vista.

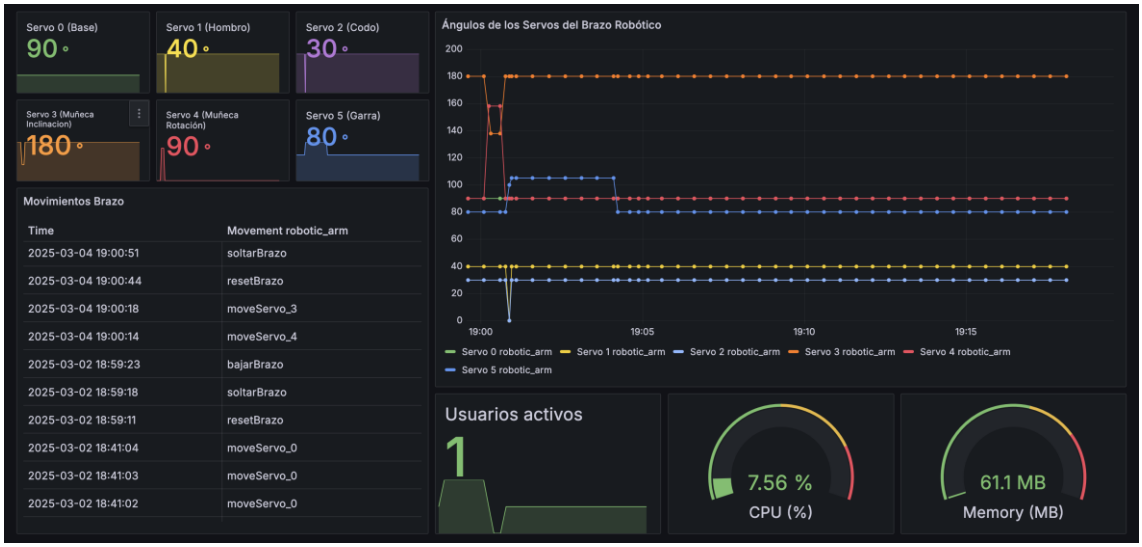


Figura 6.13: Panel de Grafana

6.2. Discusión

A través de estas pruebas se puede demostrar que el sistema cumple con todos los requisitos marcados desde el inicio del proyecto. La función más importante de este proyecto es el control y supervisión del brazo robótico de forma remota. Con la implementación de la interfaz web se crea un entorno sencillo e intuitivo de control para el usuario final de este prototipo. El control mediante las barras deslizantes permite un desplazamiento en todos los motores de manera coherente y sin latencia. Facilitando una interacción entre ambas partes del sistema de manera instantánea y sin sufrir retardo entre las peticiones y la actuación.

Por otro lado, los movimientos predefinidos generan unas secuencias que son útiles para el control del brazo robótico. Siendo accionadas de manera sencilla, a través de botones, permite realizar desplazamientos rutinarios sin necesidad de hacer uso de las barras deslizantes. Mediante estas acciones se optimiza el tiempo y precisión para realizar tareas como el agarre y el depósito de un objeto, resultando un desplazamiento autónomo sin necesidad de ser accionado individualmente.

En cuanto a la visualización de la página web, se puede observar la posición del brazo robótico en todo momento a través de su interfaz. Por otro lado, la inhabilitación de los controles permite observar que el brazo está realizando una secuencia y no permite realizar ninguna petición durante su transcurso. Una vez finalizado, los controles son habilitados según la acción realizada, permitiendo observar la posición final de cada uno de los motores.

El uso de WebSocket cumple los requisitos de generar un entorno colaborativo en el que varios usuarios pueden controlar el dispositivo al mismo tiempo. A través de la sincronización del estado de los controles, se puede observar las solicitudes expedidas desde otro dispositivo. A su vez, el uso del archivo JSON para almacenar el último estado de la interfaz resulta exitoso. Cada vez que el usuario carga la página, permite obtener la situación actual del brazo robótico, permitiendo así la reanudación del manejo sobre el brazo robótico.

Por último, el uso de la base de datos permite almacenar y monitorizar los datos más relevantes del sistema. Actuando como cliente en el broker MQTT, escucha cada uno de los mensajes publicados en dicha plataforma para su almacenaje de manera exitosa. A través de Grafana se puede observar dicha información para poder supervisar el sistema a simple vista. Estos datos permiten revisar el histórico de todos los movimientos realizados; así como, una visión general del servidor web para poder monitorizar su rendimiento.

7. Conclusiones y trabajos futuros

7.1. Conclusiones

Este Trabajo de Fin de Grado tiene como propósito final el control remoto de un sistema robótico a través de una aplicación web. Esta aplicación no solo debía tener la capacidad de accionar el brazo, sino también monitorizar su estado en todo momento. Por ello, a lo largo de su transcurso se han ido marcando los diferentes objetivos para obtener un sistema preciso e intuitivo.

Dichos objetivos se han cumplido de manera exitosa mediante la implementación de las distintas partes del sistema. A través del microcontrolador ESP8266 se ha podido procesar el intercambio de mensajes para la posterior actuación del robot. La página web, alojada en un servidor NodeJS, es capaz de realizar las peticiones definidas para mover las distintas partes del brazo. Además, se implementa una base de datos con la capacidad de monitorizar los distintos segmentos del sistema. Para el funcionamiento de los tres segmentos del sistema es crucial la comunicación entre ellos a través del servidor MQTT. Todo ello, resulta como un sistema que cumple cada una de las propuestas iniciales, permitiendo la supervisión y manipulación sobre el dispositivo hardware.

Durante este proyecto se han ido adquiriendo nuevos conocimientos que amplían los obtenidos en las asignaturas de Redes Avanzadas y Sistemas Basados en Computadores. En cuanto a las redes, gracias a este proyecto se ha entendido mucho mejor su funcionamiento y se ha ampliado el conocimiento al implementar nuevos protocolos que no han sido utilizados previamente. La parte del almacenamiento de datos de sistemas físicos ha resultado útil para asentar los conocimientos previos, haciendo uso de nuevas plataformas. Por otro lado, a través de la programación de la parte hardware se han mejorado las competencias sobre el uso de microcontroladores y el desempeño de los actuadores usados.

En conclusión, se han cumplido cada uno de los objetivos definidos al iniciar este proyecto para controlar y supervisar un sistema robótico. Además, su desarrollo ha sido enriquecedor para obtener nuevas competencias y mejorar la capacidad de resolución de problemas. Estos conocimientos serán de gran ayuda para el desarrollo de los diferentes proyectos o trabajos a los que habrá que enfrentarse a lo largo de toda la carrera profesional.

7.2. Impacto social y medioambiental

Al desarrollar un sistema de control-supervisor sobre un brazo robótico, su campo de aplicación puede ser muy diverso. El área con mayor potencial de aplicación es la industria, donde se puede automatizar la producción de diversas tareas. Debido a ello, este prototipo puede tener un gran impacto tanto a nivel social como medioambiental.

En cuanto al impacto social, atendiendo a los Objetivos de Desarrollo Sostenible, este sistema contribuye con el Objetivo 8: "Trabajo decente y crecimiento sostenible". Este sistema puede ser utilizado en naves industriales de todo el mundo, facilitando la mayoría de las tareas y mejorando la seguridad laboral. En la mayoría de las industrias se realizan trabajos que pueden suponer un riesgo para el trabajador, exponiéndole a altas temperaturas o situaciones que requieren de un esfuerzo excesivo. El control remoto de un dispositivo robótico puede suponer la mitigación de los peligros a los que se enfrenta un trabajador, obteniendo un papel de supervisor de manera remota.

Además de la mejora en la seguridad laboral, este sistema puede suponer un crecimiento en la producción de la mayoría de las empresas. La automatización de diferentes procesos representa una mejora en la eficiencia operativa suponiendo tiempos de producción más rápidos. Gracias a esto, se puede incrementar la competitividad de la empresa, adaptándose así a una alta demanda del mercado. Por tanto, su implementación supondría una gran mejora en un sistema de fabricación y manufactura, haciendo de él un entorno más seguro y rentable.

A nivel medioambiental, se persigue una reducción en el consumo de recursos al optimizar los procesos industriales. La capacidad de un sistema robótico de mantener un trabajo constante puede suponer una reducción frente al malgasto de energía. Además, el sistema está capacitado para poder funcionar a través de energías renovables, lo que supondría una disminución notoria en la huella ambiental. Por otro lado, al crear un sistema preciso se evita en gran medida la generación de residuos derivados de una mala práctica durante el proceso de una tarea.

En definitiva, la implementación de sistemas robóticos como el de este proyecto ayudan a mejorar la seguridad laboral y a crear un espacio de producción más respetuoso con el medioambiente. Al optimizar procesos y evitar riesgos se contribuye a una producción sostenible que es beneficiosa para todas las partes.

7.3. Líneas futuras

Durante el transcurso del proyecto y al finalizar con los objetivos propuestos, se han analizado todas las funcionalidades del sistema para plantear posibles líneas futuras. En este apartado se destacan las más relevantes:

- **Interfaz web:** Al realizar una aplicación web para el control y supervisión de un brazo robótico se opta por la opción más intuitiva y sencilla. Para este proyecto se ha decidido usar un control mediante las barras deslizantes y los botones proporcionando una visión general sobre el estado del dispositivo. No obstante, para hacerlo más ilustrativo se podría usar un diseño 3D del brazo. A través de este diseño se podría interactuar con cada una de las articulaciones, facilitando el desplazamiento de cada parte del brazo.
- **Prototipo robot:** Otra mejora a realizar es la implementación de un prototipo del brazo robótico más complejo. En este se podrían realizar movimientos que no fuesen tan acotados, permitiendo su desplazamiento en todas las direcciones. Además, se podrían implementar sensores los cuales estuvieran capacitados para detectar objetos. A través de estos, se mejoraría la ejecución de los movimientos predefinidos, dándole capacidad al manipulador de identificar la posición del objeto a recoger. También podrían detectar cuando ha sido agarrado un objeto para así parar el cierre de la pinza y no realizar excesiva presión sobre este.
- **Servidor:** A la hora de acceder a la página web desde un dispositivo y para realizar la comunicación entre los segmentos del sistema, es necesario que todas las máquinas estén conectadas a la misma red. Por ello, una mejora sería el desplazar el sistema a un servidor el cual pueda ser accedido de manera remota. Así, se evitaría la necesidad de acceder a través de las direcciones IP de cada uno de los segmentos, facilitando la interacción entre los dispositivos sin estar en la misma red local.

Referencias

[1] International Federation of Robotics, “Record of 4 million robots in Factories Worldwide,” IFR Press Releases, Sep. 2024. [Online]. Available: https://ifr.org/ifr-press-releases/news/record-of-4-million-robots-working-in-factories-worldwide

[2] Duplostock, “¡Robots en la fábrica! La revolución de la eficiencia en la Industria Manufacturera,” 2024. [Online]. Available: https://duplostock.com/robotica-industrial-y-colaborativa-en-la-fabricacion/

[3] Campo Digital, “Conoce a BERRY, un robot para Invernaderos de Fresas,” Apr. 22, 2024. [Online]. Available: https://campodigital.es/conoce-a-berry-un-robot-que-analiza-la-calidad-y-madurez-de-las-fresas/

[4] Word Economic Forum, “5 ways that robotics are transforming healthcare,” Sep 18, 2024. [Online]. Available: https://www.weforum.org/stories/2024/09/robots-medical-industry-healthcare/

[5] Intuitive Surgical, “Robotic-Assisted Surgery with da Vinci Systems,” [Online]. Available: https://www.intuitive.com/en-us/patients/da-vinci-robotic-surgery

[6] K. Dalamagkidis, K. P. Valavanis, and L. A. Piegl, “Aviation history and Unmanned Flight,” SpringerNatureLink [Online]. Available: https://link.springer.com/chapter/10.1007/978-94-007-2479-2_2

[7] Michael E. Moran, MD, “The da Vinci Robot,” Endourological Society, vol. 20, no. 12, Dec. 2006. [Online]. Available: https://www.endourology.org/images/endourology-history-articles/The-da-Vinci-Robot.pdf

[8] J. Alvarez, “Jacques de Vaucanson, el inventor de los primeros robots,” La Brújula Verde, Aug. 2022. [Online]. Available: https://www.labrujulaverde.com/2022/08/jacques-de-vaucanson-el-inventor-de-los-primeros-robots

[9] J. Chaves Palacios, “Desarrollo tecnológico en la primera revolución industrial,” Dehesa Universidad de Extremadura, 2004. [Online]. Available: https://dehesa.unex.es/handle/10662/10305

[10] Oxford English Dictionary, “Robot,” OED, [Online]. Available: https://www.oed.com/dictionary/robot_n2?tl=true

[11] John M. Jordan, “The Czech Play That Gave Us the Word ‘Robot,” MIT Press, Jul. 2019, [Online]. Available: https://thereader.mitpress.mit.edu/origin-word-robot-rur/

[12] Automate, “Unimate - The First Industrial Robot,” [Online]. Available: https://www.automate.org/robotics/engelberger/joseph-engelberger-unimate

[13] SRI International, “Shakey the Robot,” [Online]. Available: https://www.sri.com/hoi/shakey-the-robot/

[14] H. Makino, “Development of the SCARA,” Jan. 2014, [Online]. Available: https://www.jstage.jst.go.jp/article/jrobomech/26/1/26_5/_pdf

[15] A. L. G. Morrell et al., “The history of robotic surgery and its evolution: When illusion becomes reality,” Revista do Colegio Brasileiro de Cirurgioes, Jan. 2021, [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC10683436/

[16] Canadian Space Agency, “About Canadarm2,” [Online]. Available: https://www.asc-csa.gc.ca/eng/iss/canadarm2/about.asp

[17] Z. Chan, “Partes de un robot industrial: Componentes básicos para su funcionamiento,” British Federal Mexico, Nov. 2023, [Online]. Available: https://bfmx.com/automatizacion-industrial/partes-de-un-robot-industrial/

[18] M. Rodriguez, “¿Qué son los grados de libertad (GDL) en un robot?,” INESEM, [Online]. Available: https://www.inesem.es/revistadigital/gestion-integrada/diferencia-robotica-grados-libertad-movilidad-3/

[19] D. Ricardo, “Robots cartesianos: qué son y cómo funcionan,” Plástico, Mar. 2023, [Online]. Available: https://www.plastico.com/es/noticias/robots-cartesianos-que-son-y-como-funcionan

[20] Make Block, “Robot cilíndrico: estructura interna, funcionamiento y aplicaciones,” Jun. 2024, [Online]. Available: https://makeblock.com.ar/robot-cilindrico/

[21] Makeblock, “Brazo robótico polar: estructura interna, ventajas, desventajas y aplicaciones”, 2023, [Online]. Available: https://makeblock.com.ar/polar-robot-arm/

[22] Standard Bots, “What is a SCARA robot? A brief introduction,” Mar. 2024, [Online]. Available: https://standardbots.com/blog/what-is-a-scara-robot-a-brief-introduction?srsltid=AfmBOoq7Fo9guEoO-WNWwBNhFnIc1gzAhBao0OHAEVfxfE6pE7zAqaoo

[23] C. Bernier, “Delta Robots: The Key to Increasing Manufacturing Speed,” How To Robot, Sep. 2021, [Online]. Available: https://howtorobot.com/expert-insight/delta-robots#head1

[24] Electrón Perdido, “Brazo robot 6 grados de libertad (6dof) EXTENDIDO,” [Online]. Available: https://electronperdido.com/shop/robots/brazos/brazo-robot-6-grados-de-libertad-6dof-extendido/

[25] Electrón Perdido, “Servo MG996R,” [Online]. Available: https://electronperdido.com/shop/servos/servo-mg996r/

[26] RC Hobby Centre, “DS3115MG 15Kg Metal Geared Digital Servo,” [Online]. Available: https://rchobbycentre.co.uk/ds3115mg-15kg-metal-geared-digital-servo-1657-p.asp

[27] Espressif Systems, “EP8266 technical reference,” [Online], Available: https://www.espressif.com/sites/default/files/documentation/esp8266-technical_reference_en.pdf

[28] Grafana Labs, “Get started with Grafana and InfluxDB”, [Online], Available: https://grafana.com/docs/grafana/latest/getting-started/get-started-grafana-influxdb/

Anexos

El código desarrollado para este Trabajo de Fin de Grado se puede encontrar en los siguientes repositorios:

- Microcontrolador: https://github.com/GuillermoCarrascoRoncero/TFG_Arduino
- Servidor web: https://github.com/GuillermoCarrascoRoncero/TFG_ServidorWeb
- Base de datos: https://github.com/GuillermoCarrascoRoncero/TFG_InfluxDB

Además, estos códigos vienen almacenados en el archivo comprimido utilizado para la entrega de esta memoria.