UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de un programa de control para una máquina de ensayos biaxial

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Ros Briones, Hernán

Tutor/a: Sánchez López, Miguel

CURSO ACADÉMICO: 2022/2023

Resumen

El trabajo se basa en la creación de un controlador para una máquina de ensayos biaxial. La funcionalidad y el control de ésta se desarrolla mediante un microcontrolador que trabaja en el lenguaje MicroPython utilizando el framework de Microdot, y para visualizar y modificar sus datos se utiliza una interfaz web desarrollada con HTML, CSS y JavaScript.

Palabras clave: microcontrolador, interfaz, wifi, MicroPython, JavaScript, HTML, ensayos, máquina, simulación

Abstract

The project is based on the creation of a controller for a biaxial testing machine. Its functionality and control are developed using a microcontroller that operates on the MicroPython language using the Microdot framework, and to visualize and modify its data it employs a web interface that is developed with HTML, CSS and JavaScript.

Key words: microcontroller, interface, wifi, MicroPython, JavaScript, HTML, testing, machine, simulation

Resum

El treball es basa en la creació d'un controlador per a una màquina d'assajos biaxial. La funcionalitat i el control d'aquesta es desenvolupa mitjantçant un microcontrolador que treballa amb el llenguaje MicroPython utilitzant el framework de Microdot, i per a visualitzar i modificar les seues dades s'utiliza una interfície web desenvolupada amb HTML, CSS i JavaScript.

Paraules clau: microcontrolador, interfície, wifi, MicroPython, JavaScript, HTML, assajos, màquina, simulació

Índice de contenidos

1. Introducción

Durante este proyecto, vamos a exponer de forma detallada el desarrollo de un controlador para una máquina de ensayos biaxial usando el lenguaje MicroPython. Dicho controlador se va a implementar para ser utilizado mediante el uso de clientes web para así acceder de forma remota a la interfaz de la máquina. A continuación, vamos a detallar la motivación para realizar este trabajo, los objetivos que pretendemos alcanzar con éste, las alternativas actuales respecto a las de este proyecto y muchas otras secciones que corresponden con la planificación, creación y testeo del mismo.

1.1. Motivación

En la actualidad, el desarrollo de las TIC ha permitido que muchas pequeñas y medianas empresas, que anteriormente se habían mantenido siempre bastante alejadas de las nuevas tecnologías, puedan disponer de ellas para impulsar sus negocios. Sin embargo, estas mejoras todavía no se puede decir que estén generalizadas y aún queda un gran trabajo que hacer para extenderlas a la mayor parte de negocios. Una buena forma de hacerlo es facilitar el uso de estas tecnologías a los emprendedores para animarlos a digitalizar sus proyectos y así agilizar su carga de trabajo.

Por esta razón, vamos a desarrollar para una máquina de ensayos biaxial su código de controlador y una interfaz gráfica accesible desde un cliente web para ayudar a los usuarios artesanos a monitorizar en todo momento los ensayos que se realizan. De esa forma, no es necesario para el beneficiario disponer de un solo dispositivo para realizar las pruebas con una aplicación de escritorio, sino que se puede utilizar cualquier dispositivo que se pueda conectar a la misma red Wifi a la que se conecte el controlador. Además, incluso se permite que el control se realice desde más de un cliente simultáneamente.

En cuanto a mi motivación personal, me interesa el desarrollo de este proyecto ya que siempre he querido saber como manejarme con la tecnología de microcontroladores. Nunca he tenido experiencia con ellos y este trabajo me da la oportunidad de conocer su funcionamiento. Esto abre la puerta a que pueda desarrollar futuros proyectos de carácter más profesional, o también más personal como aprender a controlar dispositivos dentro de mi hogar a mi gusto.

Además, puedo mejorar mis habilidades para la creación de una interfaz web que es algo que no he tenido la oportunidad de ver en la carrera. De esta forma, estaría familiarizándome con lenguajes como HTML, CSS y JavaScript que resultan esenciales para manejarse en el mundo laboral de la informática.

1.2. Objetivos

Para poder alcanzar las expectativas mencionadas anteriormente, definimos los siguientes objetivos principales que posteriormente vamos a desarrollar en detalle para poder orientar de forma más adecuada y metódica la realización del proyecto:

1. Adquirir un buen conocimiento sobre los lenguajes altamente utilizados de Python, HTML, CSS y JavaScript. Esto da pie a poder estudiar MicroPython para microcontroladores y como diseñar interfaces web.

2. Crear el firmware de un microcontrolador Raspberry Pi Pico W utilizando el lenguaje de MicroPython para controlar las señales de entrada y salida de la máquina de ensayos biaxial.
3. Desarrollar una interfaz web de usuario intuitiva y sencilla de manejar por parte del cliente que pueda ser accedida desde cualquier dispositivo mediante una conexión inalámbrica a Internet.

4. Mejorar mis habilidades de resolución de problemas complejos, mi destreza con los entornos de desarrollo y mi creación de interfaces webs y manejadores.

1.3. Metodología

Una vez definidas las metas a alcanzar en el proyecto podemos profundizar en los pasos que hay que efectuar para poder alcanzarlos de forma sistemática. Vamos a desarrollar los pasos según los objetivos anteriores:

<u>Documentación previa</u>

- Descargar y familiarizarse con el entorno de desarrollo que utilicemos en el proyecto. En este caso, el elegido ha sido Thonny ya que está explícitamente diseñado para trabajar con lenguaje Python y MicroPython.

- Adquirir los conocimientos básicos del lenguaje Python como las variables, tipos de datos, operadores, estructuras de control, funciones, etc.

- Estudiar las librerías y módulos básicos de Python como los utilizados para trabajar con operaciones, fechas y horas, manejo de archivos, etc. Además aprender las librerías específicas empleadas en nuestro microcontrolador.

- Aprender las características y diferencias entre Python y MicroPython (que es con lo que vamos a programar), así como también la programación de entradas y salidas con sus respectivas librerías.

- Aprender los aspectos básicos de HTML, CSS y JavaScript y cómo funcionan en conjunto para ofrecernos la funcionalidad de una página web.

- Analizar y comprender proyectos reales encontrados mediante recursos en línea para ver cómo se implementan los conocimientos adquiridos mencionados antes.

Desarrollo del firmware del microcontrolador

- Explorar la documentación oficial del microcontrolador para aprender su estructura, características y rendimiento.

- Configurar el entorno de desarrollo utilizado para ser compatible con el lenguaje y el microcontrolador.

- Concretar las funcionalidades y métodos a implementar posteriormente en el firmware teniendo en cuenta las limitaciones del dispositivo.

- Desarrollar el código de las funcionalidades mencionadas en el punto anterior de forma modular y sistemática en subapartados. Antes de pasar al siguiente subapartado debemos depurar y testear el código para comprobar su correcto funcionamiento.

- Revisar el código entero una vez terminado para considerar si puede ser optimizado.

Desarrollo de la interfaz de usuario

- Crear el diseño base visual de la interfaz en HTML y definir la funcionalidad de los manejadores que va a ofrecer nuestro código en JavaScript.

- Desarrollar una conexión web dentro del microcontrolador que pueda servir como punto de acceso de los clientes web para poder acceder a la interfaz y mostrarla en el navegador.

- Implementar la funcionalidad de la interfaz e integrarla con el servidor web del microcontrolador para que cada vez que se produzcan cambios en la placa o en la interfaz el otro elemento reciba un mensaje para reaccionar como debe.

- Depurar y probar el código para asegurar un correcto funcionamiento al recibir solicitudes de clientes web y enviar sus respectivas respuestas, así como probar el código cuando la máquina de ensayos envíe datos a la interfaz.

- Revisar y optimizar el código para mejorar el rendimiento de las conexiones.

1.4. Estructura

La estructura de todo el proyecto va a desarrollar de forma detallada los siguientes apartados:

1. Estado del arte: Se recopila información sobre el entorno del proyecto a desarrollar. Se detalla si existen otras alternativas en el mercado o similares.

2. Análisis del problema: Utilizando las alternativas del estado del arte se estudia cómo se puede obtener una propuesta superior a las anteriores. Analizando las diferentes propuestas se decide cuál vamos a desarrollar en este TFG.

3. Diseño de la solución: Dicha propuesta se describe detalladamente y se describen sus componentes físicos. Después se procede a decidir cómo va a ser su arquitectura y su diseño detallado.

4. Desarrollo de la solución: Se especifica la evolución del proyecto desde el inicio hasta el resultado final, comentando los obstáculos que han ido surgiendo y los cambios que se han tenido que adoptar.

5. Pruebas: Se testea el funcionamiento final del proyecto y se analiza para comprobar si realiza lo esperado o si son necesarios cambios para optimizar el rendimiento.

6. Conclusiones: Se aclara si se han alcanzado los objetivos declarados previamente o no. También se debe aclarar si han sido útiles los conocimientos adquiridos en la carrera y reflexionar sobre la capacidad de resolución adquirida al finalizar el proyecto.

2. Estado del arte

En este apartado vamos a estudiar el “contexto tecnológico” de nuestro trabajo. Además de describir el concepto de máquina de ensayos biaxial, estudiaremos las diferentes propuestas que hay en el mercado y sus características.

2.1. Concepto de máquina de ensayos biaxial

Las máquinas de ensayos biaxiales son dispositivos utilizados para comprobar el comportamiento de estructuras y materiales bajo distintos tipos de fuerza en diferentes ejes. Son ampliamente usadas en multitud de campos de ingeniería e investigación ya que permiten saber con mucha precisión la condición, las características y el comportamiento mecánico de los materiales en tiempo real.

Estas máquinas son capaces de aplicar de formas muy diversas cargas a los materiales a testear, pero en concreto las biaxiales tienen la capacidad de ejercer fuerzas de tracción, corte, torsión, compresión y combinaciones de éstas en diferentes direcciones. Esto simula de forma ideal cualquier situación que pueda darse en un entorno práctico real. Algunas de las industrias dónde se incluye esta tecnología pueden ser:

- Investigación de materiales: Aquí se evalúa todo tipo de características de sus productos, lo que es fundamental para la creación de materiales nuevos para aplicaciones específicas.

- Construcción e ingeniería civil: Se utiliza para realizar pruebas con gran relevancia ya que se examina la resistencia y el comportamiento de estructuras como edificios, puentes, etc.

- Investigación biomecánica: Aquí se realizan pruebas más sofisticadas debido a la fragilidad de los materiales como tejidos biológicos, prótesis, huesos, tendones, ligamentos, etc.

En general, las máquinas de ensayos biaxiales desempeñan una labor muy importante a la hora de mejorar la comprensión del comportamiento de materiales de todo tipo. Incluso para nuestro proyecto, la utilidad de estas máquinas para los usuarios artesanos es de vital importancia para el crecimiento de sus negocios y la mejora de sus técnicas.

2.2. Propuestas en el mercado

Los productos que se comercializan en el mercado se pueden hallar en diferentes empresas que se dedican a la fabricación y venta de las mismas. Se pueden hallar bastantes modelos con diferentes modalidades dependiendo del tipo de materiales o pruebas que requiera tu propuesta. Existen diferentes modelos que varían en la fuerza máxima ejercida sobre los materiales dependiendo de la cantidad de precisión que deseas desempeñar. Unas de las opciones que podemos hallar con funcionalidades similares a las que buscamos en este proyecto son las siguientes:

- ZwichRoell (Máquina de ensayos cruciforme): Esta empresa ofrece diferentes modelos de máquina de ensayos biaxial en función de la carga que puede ejercer a los materiales (hasta 250kN). En nuestro caso, tan solo es necesario un modelo con una fuerza máxima de 2kN. Éste ofrece un dispositivo de 4 ejes controlados por actuadores independientes que controlan los pistones. Además, el cliente puede realizar ensayos personalizados ya que incluye un programa editor gráfico.

Figura 1. Máquina de ensayos cruciforme de la compañía ZwichRoell

- MTS (Sistema de pruebas biaxial planar): MTS ofrece también varios modelos de máquinas de ensayos biaxiales hasta los 500kN de fuerza máxima ejercida. Vamos a fijarnos en el de 25kN ya que no hay otro con menor valor. MTS nos ofrece un producto con pistones servo hidráulicos capaces de testear materiales estáticos y dinámicos manteniendo un control preciso con sus herramientas de monitoreo. Asimismo, simula las fuerzas ejercidas en un entorno real o uno ofrecido por el cliente, además de poder simular condiciones de elevada temperatura.

Figura 2. Sistema de pruebas biaxial planar de la compañía MTS

- TA Instruments (Planar biaxial TestBench instrument): Esta empresa ofrece una máquina de ensayos biaxial más modesta que las anteriores en lo que respecta a fuerza ejercida (200N), sin embargo ofrece una alta precisión en los test de materiales. Además, ofrece un software de monitoreo que puede controlar parámetros específicos que elija el cliente como el desplazamiento, la tensión o la carga. Incluso puede aplicar fuerzas de formas más sofisticadas para poder simular ondas.

Figura 3. Planar biaxial TestBench Instrument de la compañía TA Instruments

- Admet (eXpert 8000 planar biaxial testing machine): Admet propone un modelo de máquina de ensayos biaxial de hasta 5kN que se puede complementar con una gran variedad de complementos que ofrecen distintos tipos de agarre. También es capaz de realizar pruebas bajo líquidos y simular temperaturas elevadas.

Figura 4. eXpert 8000 planar biaxial testing machine de la compañía Admet

3. Análisis del problema

Después de analizar detenidamente las propuestas comerciales similares a nuestro proyecto podemos comenzar a destacar aquellas propiedades que sean útiles para nuestro trabajo y otras que se puedan mejorar.

El principal problema de los productos comentados en el estado del arte es lo limitado que está el software controlador de sus dispositivos. Tan solo se ofrece una interfaz de usuario mediante una aplicación de escritorio la cual necesita que el controlador de la máquina de ensayos esté conectado mediante un puerto serial. Esto limita mucho el control del dispositivo ya que está ligado a un solo lugar. Además, aparte de ser aplicaciones de escritorio, este software siempre se trata de software de usuario ya preestablecido. Esto restringe en gran medida la personalización del mismo respecto de la máquina particular con la que se está trabajando. Así mediante una interfaz estándar para todos los usuarios que compren ese producto están consiguiendo un rendimiento del software aceptable, pero en muchas ocasiones no es óptimo para cada máquina de ensayos biaxial.

A pesar de los aspectos a mejorar de las propuestas anteriores, cabe destacar que hay ciertos aspectos interesantes de éstas que valoramos implementar a nuestro proyecto. Una de ellas es la inclusión de una gráfica para visualizar cómodamente cómo se comportan los materiales a tiempo real, lo cual va a resultar esencial para que el cliente final disfrute de una interfaz de calidad. También hay que recalcar la utilidad de la opción de poder mover los motores de los ejes de forma independiente de los demás, además de poder apagar o encender los motores a voluntad según queramos utilizarlos o no.

Después de distinguir los puntos a mejorar y destacar ya podemos crear una solución para nuestro proyecto que solvente los puntos que hemos comentado. En este caso no va a incluir la creación de la totalidad de la máquina de ensayos sino que vamos a centrarnos en establecer el software que la controla, o lo que es lo mismo, el firmware del microcontrolador y la interfaz web de usuario. En gran medida nos concentramos en este campo ya que, como hemos comentado hace dos párrafos, todos los aspectos a mejorar tienen relación con el software controlador de la máquina y no tanto con el hardware de ésta.

La solución a este problema que proponemos en nuestro proyecto es crear dentro de un microcontrolador un servidor web que podrá ser accedido por los clientes desde un navegador con conexión a internet. De esta forma, el microcontrolador que estará conectado a la máquina de ensayos mediante sus pines de entrada/salida compartirá cualquier dato que le llegue para que lo pueda ver el cliente en tiempo real. Y a su vez también recibirá instrucciones desde la interfaz para enviarlas a la máquina. Esto mejora enormemente la

versatilidad del software de control para que pueda ser mucho más intuitivo y cómodo de utilizar. Si a eso le sumamos que lo vamos a programar para que sea de código abierto, se añade un gran potencial de mejora para progresar en el uso de esta tecnología y que pueda ganar relevancia en el mercado actual, ya que se podrá utilizar el software diseñado como base para adaptarse a las necesidades de cualquier otra máquina de ensayos. Asimismo la interfaz será diseñada con campos para enviar datos hacia el microcontrolador, para mostrar los datos que llegan desde la máquina de ensayos tanto numéricamente como gráficamente y la posibilidad de apagar o encender los motores.

Una vez diseñada la solución final para el proyecto pasamos a tener que ponerla en práctica mediante la tecnología que tenemos a nuestra disposición. Para ello tenemos que encontrar aquellos componentes y tecnología necesarios para implementar la solución definitiva a continuación.

4. Diseño de la solución

En este apartado vamos a llevar a cabo la descripción de los componentes físicos del sistema y a detallar cuál va a ser la arquitectura y el diseño del proyecto.

4.1. Componentes físicos

Nuestra solución final para el proyecto pese a estar solo enfocada al desarrollo del software controlador de la máquina de ensayos biaxial necesita de un elemento físico esencial para interactuar como intermediario entre el cliente y la máquina.

4.1.1. Microcontrolador

Para la propuesta del microcontrolador hemos elegido trabajar con un dispositivo de la compañía Raspberry Pi Foundation, en concreto el dispositivo Raspberry Pi Pico W. Se trata de uno bastante versátil y compacto que se adapta a infinidad de tareas a la vez que tiene bastante potencia sin consumir mucha energía. Es muy similar al Raspberry Pi Pico, su modelo predecesor, pero con conectividad inalámbrica como añadido.

Este microcontrolador cuenta con un chip RP2040 que integra un procesador de doble núcleo ARM Cortex-M0+ de 32 bits que trabaja a 133MHz. Para poder interactuar con periféricos del entorno contiene 26 entradas/salidas con propósito general (GPIO) multifunción y otras 23 solo digitales de las cuáles 3 permiten conversión de señal analógica a digital (ADC). Estos pines ofrecen una conexión con una gran variedad de sensores, actuadores y muchos otros componentes electrónicos de forma rápida y programable por el usuario.

Respecto a su memoria, la Raspberry Pi Pico W cuenta con 2MB de memoria flash para poder almacenar datos y código. Además, su memoria SRAM es de 264KB partida en 6 bancos para poder acceder de forma simultánea a la memoria. Ésta es muy útil para manejar aplicaciones sencillas con bastante rendimiento.

Las conexiones con el microcontrolador se pueden realizar de la forma tradicional mediante los pines para la comunicación en serie (UART), los pines para la conexión en bus (I2C y SPI) y los canales para PWM. También cuenta con un conector micro-USB para poder alimentar la placa, programar la funcionalidad de ésta y para proveerla con pulsos de reloj. Es posible además alimentar la placa sin necesidad del micro-USB a través de pines GPIO específicos. Además una de las conexiones a destacar es su módulo wifi integrado para la transmisión y recepción de señales inalámbricas para poder utilizar servicios y aplicaciones basadas en red.

En cuanto a la programación del dispositivo se ofrecen varias alternativas de lenguajes de programación para poder hacer más accesible la tarea para el desarrollador. Puede ser programada en una versión específica de Python diseñada para trabajar con microcontroladores denominada MicroPython. Este va a ser el lenguaje elegido para el proyecto. De igual modo también es posible trabajar con C/C++ utilizando el kit de desarrollo de software (SDK) que ofrece Raspberry Pi Fundation, pero también es compatible con otras plataformas de desarrollo como puede ser Arduino.

Con todas estas funciones podemos concluir que el microcontrolador Raspberry Pi Pico W es una opción más que acertada para poder llevar a cabo el desarrollo del proyecto sin que haya problemas de rendimiento o capacidad.

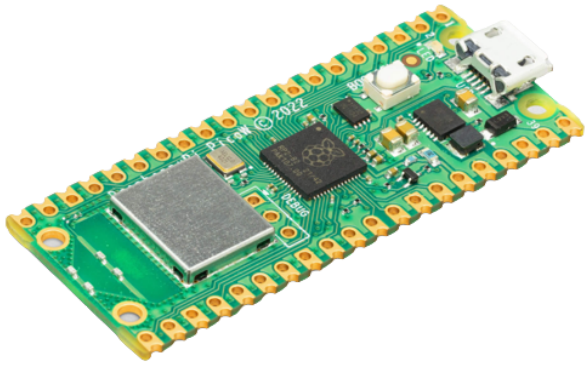

Figura 5. Microcontrolador Raspberry Pi Pico W

4.2. Arquitectura del sistema

Es importante antes de comenzar a desarrollar código o cualquier función específica del proyecto, centrarnos primero en concretar cómo van a estar distribuidos los componentes y cómo interactúan entre sí. Así establecemos una base sólida que resulta fundamental para el desarrollo posterior del software a implementar y evitamos tener que repetir o descartar trabajo ya realizado por falta de coherencia en el conjunto de los elementos finales del proyecto.

Vamos a dividir nuestro sistema en dos capas, la capa de presentación y la capa de negocio. Como ya hemos estudiado a lo largo de la carrera, en la capa de negocio nos encargamos de implementar las funcionalidades de nuestro sistema para procesar los datos de la máquina y realizar cálculos. En ella vamos a trabajar con el firmware del microcontrolador. Éste se comunica con la capa de presentación que se encarga de la interacción con los usuarios y conseguir que la visualización de los datos sea comprensible e intuitiva. En ésta va a estar la interfaz de usuario. Esta separación en capas facilita el posterior mantenimiento del sistema y su escalabilidad.

La conexión entre la máquina de ensayos y el microcontrolador está diseñada para ser establecida mediante los pines de conexión de entrada/salida, pero en este caso no va a estar implementada sino que la vamos a simular. Sin embargo, la conexión entre el microcontrolador y la interfaz va a ser inalámbrica. Este hecho abre la posibilidad de aumentar la escalabilidad del sistema al admitir varios enlaces simultáneos. Para aprovechar esta cualidad del sistema vamos a optar por implementar un modelo de cliente-servidor en el cual en el microcontrolador encontraremos el servidor al que se pueden conectar múltiples clientes web.

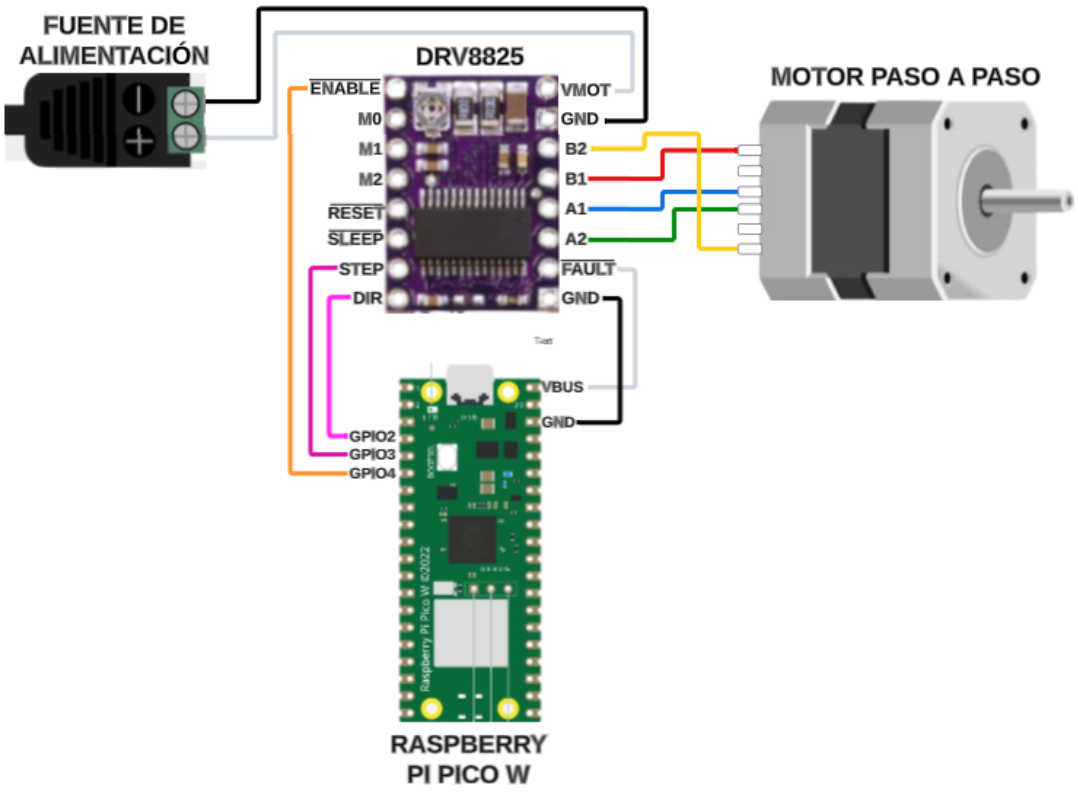

Figura 6. Diagrama de conexión entre microcontrolador y un motor paso a paso de la máquina de ensayos (representación ficticia de posible máquina de ensayos real)

Antes de la realización del proyecto hemos estudiado y seleccionado de antemano las plataformas utilizadas para desarrollar el código. Hemos decidido trabajar con el lenguaje de programación MicroPython para la placa ya que se trata de un lenguaje muy intuitivo que posee una gran cantidad de librerías y módulos con las que ofrece más sencillez a la hora de programar. También vamos a utilizar el entorno de desarrollo de Thonny porque se concibió para enfocarse en el lenguaje Python y además ofrece varias utilidades orientadas a desarrollar código de microcontrolador. Para el diseño de la interfaz web no hemos recurrido a ninguna plataforma de desarrollo de interfaces ya que es relativamente asequible de implementar a mano en HTML y CSS. El comportamiento de la misma con JavaScript vamos a implementarlo también con el entorno de desarrollo de Thonny.

Una vez decidida la arquitectura y los componentes que formarán el diseño final del trabajo ya podemos representarlos gráficamente a continuación. Esta estructura es muy similar a la que acostumbramos a ver en modelos web de cliente-servidor.

Figura 7. Diagrama de bloques de la arquitectura del proyecto

4.3. Diseño detallado

Aquí desarrollaremos en detalle cómo haremos funcionar los elementos mencionados anteriormente antes de ponernos a implementar el código de los mismos. Hemos comentado que vamos a dividir el código en capas, así que a continuación analizamos las partes que van a constituirlas.

4.3.1. Código del microcontrolador

En la capa de negocio es donde vamos a desarrollar el firmware del microcontrolador. Aquí es dónde se gestiona la lógica principal del proyecto y donde se manipulan y operan los datos provenientes de la máquina de ensayos o la interfaz web para que los datos resultantes lleguen de forma correcta a su respectivo destino. En esta capa tan solo se encuentra el código del microcontrolador.

Este elemento actuará como el intermediario entre la máquina de ensayos y el cliente para que éste tenga una sensación de transparencia. Dentro de nuestro proyecto funcionará como el servidor que da servicio a nuestra aplicación web para que los clientes web puedan tener control sobre el hardware.

Primeramente, tenemos que declarar los elementos esenciales en forma de variables para poder acceder a ellos en el código y así leerlos o modificarlos según convenga. La placa al estar conectada a la máquina mediante pines de entrada/salida (GPIO) estos deben estar declarados individualmente. En nuestro hardware disponemos de cuatro motores actuadores paso a paso que sirven para mover los brazos cada uno en dos posibles direcciones. Dicho eso, hay que declarar cuatro variables para los pines de salida correspondientes a los pasos a mover los actuadores, cuatro variables para los pines de salida correspondientes a la dirección que toma cada motor y cuatro variables para los pines de salida correspondientes al estado del motor (apagado o

encendido). Además es necesario declarar cuatro variables para los pines de entrada que corresponden con el valor de los sensores de fuerza de la máquina.

Una vez definimos los elementos a tener en cuenta pasamos a indicar la funcionalidad de la aplicación en forma de métodos. Vamos a explicar tan solo a grandes rasgos el cometido de cada método sin explicar cómo se implementan.

Como hemos hablado en este epígrafe, el servidor estará conectado al hardware mediante cables en los diferentes pines así que la conexión se realiza directamente cuando declaramos las variables anteriores. Podremos comunicar estos dos elementos directamente enviando o recibiendo señales en éstas. Sin embargo, entre el servidor y el cliente web este proceso no es tan simple. Al ser una conexión mediante wifi debemos definir un método de conexión que haga que el microcontrolador se conecte con el SSID y contraseña a la red wifi utilizada para que pueda recibir datos a través de su ip y un puerto establecido.

Después, debido al carácter de la aplicación que vamos a desarrollar, el código servidor necesita de un método que se mantenga en ejecución en todo momento para poder manejar las peticiones del cliente. Para ello es necesario que en éste haya un bucle infinito que espere a recibir solicitudes del cliente y que procese los datos antes de volver a la siguiente iteración. De esta forma, el programa nunca termina hasta que el microcontrolador se quede sin energía o que ocurra un fallo en la máquina de ensayos.

Además de esto, son necesarios aquellos métodos que al iniciar la aplicación envíen las plantillas HTML, CSS y JavaScript a los clientes web que establezcan una conexión con el servidor.

4.3.2. Plantilla HTML

Esta plantilla, al igual que el código JavaScript el cuál comentaremos en el apartado 4.3.3, forma parte de la capa de presentación. Ésta se encarga de la representación y el formato de los datos para que los usuarios clientes puedan interactuar con ellos de forma fácil e intuitiva.

Este elemento actuará como el esqueleto de la interfaz web que verá el cliente. Al ser el único elemento que va a visualizar el usuario deberá estar compuesto por todos los elementos de control necesarios para un tener un dominio adecuado de la máquina de ensayos.

Para cubrir las acciones de control necesitaremos varios componentes. Centrándonos en la acción de apagar y encender los motores añadiremos

cuatro botones, uno por cada motor. Cuando el cliente clique en el botón correspondiente al apagado de uno de los motores, éste cambiará para convertirse en el botón de encendido y viceversa. Esta forma de hacerlo es mucho más conveniente que añadir ocho botones debido a que complicaría el código y sobretodo recargaría demasiado la interfaz.

Para la acción de mover los cuatro brazos de la máquina a cualquier posición es necesario un elemento más flexible que un botón al que podamos añadir directamente un valor numérico. Es posible añadir un campo de texto donde escribir ese número pero para poder probar el comportamiento de la máquina con diferentes valores en un corto periodo de tiempo no resulta muy eficiente. Es una mejor opción agregar un deslizador (slider) para cada brazo, que se pueda clicar o incluso arrastrar que haga mucho más rápido y efectivo el control de la máquina.

Para la lectura de datos provenientes de la máquina de ensayos hay que añadir al menos cuatro campos para leer cada valor de la fuerza ejercida por los brazos a tiempo real. Esto significa que cada cierto periodo, el valor de estos campos cambiará del valor antiguo al nuevo valor. Si dicho periodo es muy pequeño el valor cambia demasiado rápido y ello no resulta en un método muy intuitivo para el cliente. Para solucionar esto se puede añadir además una representación visual de estos datos a tiempo real en una gráfica de líneas. De esta forma, podemos a simple vista tener una mejor idea del comportamiento de un material ante las fuerzas ejercidas por la máquina.

4.3.3. Código JavaScript

En el código de JavaScript va a desarrollarse el comportamiento de los elementos de la plantilla HTML descritos en el punto 4.3.2. Esto significa que cada vez que la interfaz sea modificada por el cliente, este código se encargará de cambiar su aspecto en consecuencia y notificar de los cambios al servidor del microcontrolador.

Para ello añadimos “listeners” a cada uno de los elementos modificables de la interfaz como los cuatro sliders y los botones de alimentación. En el caso de los sliders tan solo se envían hacia el servidor los datos correspondientes al valor y el identificador del brazo, mientras que los botones de corriente además de lo anterior deben cambiar la visualización de su interfaz para pasar de “apagado” a “encendido” o viceversa. Además hay que tener en cuenta que si vuelven a ser accionados su valor enviado no puede ser igual, para que así el servidor obtenga la información de forma veraz. Es importante que el código JavaScript de estos elementos tan solo se encargue de enviar los datos producidos hacia el servidor ya que así centramos en la capa de negocio todo el código referente a las operaciones sobre éstos y quedará una arquitectura mucho más clara y escalable.

Por último, para los campos de lectura de valores de la fuerza ejercida y su representación gráfica hemos de solicitar periódicamente estos valores al servidor. El ejemplo de los “listeners” aquí no resulta muy conveniente ya que estos valores en la máquina de ensayos se modifican en ciclos tan rápidos que resulta inviable verlos en tiempo real. La mejor alternativa es añadir al código un bucle donde cada cierto tiempo se soliciten estos datos de lectura al servidor y se muestren en los campos correspondientes.

5. Desarrollo de la solución

En este apartado, ya vamos a describir cómo ha sido el desarrollo de la solución propuesta una vez ya terminado el código de la misma al completo. Se analizarán uno por uno los problemas hallados y sus respectivas soluciones desde el momento de comenzar el proyecto hasta obtener la solución final.

5.1. Diseño de la interfaz (HTML)

Hemos decidido primero diseñar como verá el cliente en el navegador la interfaz web y a partir de los elementos definidos en ese HTML desarrollaremos la estructura de control para cada componente. Debido al desconocimiento de cómo subdividir el proyecto en capas traté de realizar un método en el servidor que devolviera un código HTML. Dentro de este HTML se encontraba detallado explícitamente dentro de un campo “<script>” todo su código JavaScript. El método hubiera sido enviado al cliente web como respuesta a su primera petición. Sin embargo, claramente esto hacía que la arquitectura en capas no estuviera aplicada en absoluto y que el código del servidor se hiciera muy extenso y cargante. Incluso había ocasiones en las que al ejecutar el código, éste daba problemas.

Por esas razones en un punto del desarrollo del proyecto se decidió cambiar la metodología y realizar estos pasos utilizando el framework de Microdot. Este framework está inspirado en Flask pero con un enfoque más minimalista para así funcionar en sistemas con recursos limitados como un microcontrolador y que trabaja con Python o MicroPython. De este modo, el código HTML puede estar separado del código del servidor al igual que el código CSS y JavaScript como veremos posteriormente.

Después, como ya mencionamos en el apartado 4.3, construimos una interfaz con cuatro botones para el apagado y encendido de los motores actuadores, cuatro deslizadores para el control del movimiento y dirección de los brazos, cuatro campos de muestreo para los valores de fuerza ejercidos por cada brazo y una gráfica de líneas para representar estos datos. La creación de todos estos elementos aunque en un principio no tengan funcionalidad pueden verse con claridad al abrir el documento HTML en un navegador. Todos menos la gráfica ya que es un elemento que se va a crear dinámicamente según vayan llegando datos provenientes de la máquina de ensayos. Por esta razón, construimos una región rectangular que actúa como contenedor de la gráfica de líneas con el campo “<canvas>”.

Para finalizar el HTML hemos agregado los metadatos y referencias a recursos externos necesarios para su visualización y buen funcionamiento en un navegador. En cuanto a los metadatos nos hemos centrado en lo estándar para no complicar el código. En estos se incluirían la codificación de carácteres a

UTF-8, la configuración de la vista del contenido para dispositivos móviles y la configuración de la compatibilidad con versiones antiguas de navegadores. En cuanto a la referencia a recursos externos hay que enlazar los archivos CSS y JavaScript al HTML para que funcionen correctamente. El CSS al ser una hoja de estilos se enlaza con el campo “<link>”, mientras que se usa el campo “<script>” para el código JavaScript. También añadimos las bibliotecas de JavaScript “chart.js” utilizada para la visualización interactiva de gráficos y “jquery.min.js” utilizada para manipulación de elementos HTML y el manejo de eventos.

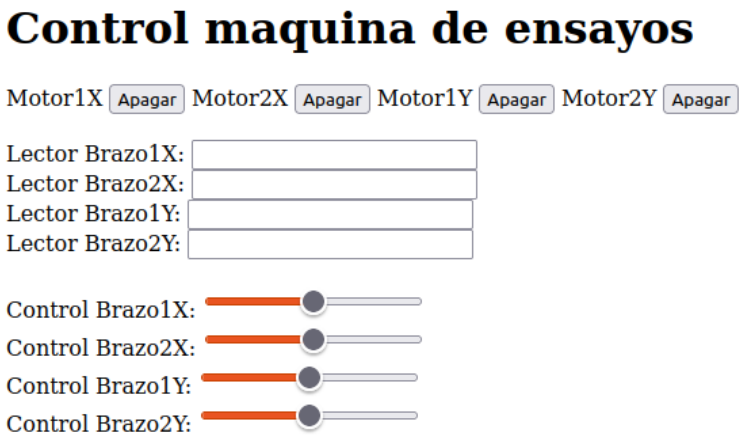

Figura 8. Plantilla HTML de la interfaz gráfica

5.2. Estilo de la interfaz (CSS)

Al ser una interfaz que se almacena dentro del microcontrolador es más conveniente mantener la hoja de estilos CSS aseada ya que queremos mantener lo más simple y fácil su visualización. No recargar mucho este elemento ayuda a centrar la atención del cliente en los datos, además de evitar que el código ocupe demasiado y así ralentizar las prestaciones del dispositivo. Es por eso que el único elemento modificado ligeramente son los botones de alimentación de los motores. Tan solo se han modificado sus medidas para agrandarlos y por defecto los hemos creado de color rojo ya que tendrán el texto “Apagar”.

Figura 9. Hoja de estilo CSS de la interfaz gráfica

5.3. Funcionalidad de la interfaz (JavaScript)

Para controlar la funcionalidad de cada elemento creado en el HTML hemos desarrollado el archivo JavaScript. Al comienzo, cómo ya se ha comentado en el punto 5.1, éste código se encontraba incluido dentro del HTML que a su vez se creaba en un método del código del servidor. Al hacer esta separación en capas, la ventaja es notable no solo para que el código sea más legible sino para que sea más sencillo de programar. Al usar lenguaje MicroPython para el servidor y JavaScript para la interfaz, nos hemos ahorrado muchas correcciones por estar escribiendo en dos lenguajes al mismo tiempo dentro del mismo archivo.

Para comenzar a programar los diferentes elementos hemos tenido que referenciarlos en variables. Esto se hace mediante el método "querySelector()" que junto con el id del elemento HTML hace que podamos trabajar la funcionalidad de éstas. También otro elemento a declarar sería la gráfica de líneas para muestrear los datos recibidos. Hemos utilizado el constructor de “chart.js” para crearla y situarla dentro del espacio rectangular dedicado para ella. Para realizar esta fusión de los dos elementos necesitamos ofrecer el contexto del “<canvas>” a la gráfica. Recordamos que el "<canvas>" actuaba como lienzo contenedor de la gráfica y tenía las dimensiones que hemos declarado anteriormente en el HTML. Para este elemento hemos indicado que fuera de líneas, que no tuviera etiquetas en el eje X y hemos creado los datasets que se representan en el gráfico. Cada dataset se ha dedicado a representar el valor de cada uno de los brazos de la máquina por lo que hemos establecido un total de cuatro. En cada dataset hemos incluido una etiqueta para identificarlos, el array para sus datos y otros elementos estéticos para diferenciarlos entre ellos. Para terminar hemos configurado que por defecto el eje Y comience en 0.

```
var chart = new Chart(ctx, {
      type: 'line',
      data: {
        labels: [], // Etiquetas para el eje x
        datasets: [
          {
            label: 'Brazo1X',
            data: [], // Datos para el eje y
            backgroundColor: 'rgba(0, 123, 255, 0.5)',
            borderColor: 'rgba(0, 123, 255, 1)',
            borderWidth: 1,
            fill: 'origin',
            lineTension: 0
          },
          {
            label: 'Brazo2X',
            data: [], // Datos para el eje y
            backgroundColor: 'rgba(255, 0, 0, 0.5)',
            borderColor: 'rgba(255, 0, 0, 1)',
            borderWidth: 1,
            fill: 'origin',
            lineTension: 0
          },
```

Figura 10. Ejemplo de dataset dentro de la gráfica

Antes de comenzar a hablar de los manejadores de eventos ligados a cada input de los elementos HTML tenemos que comentar la tecnología implementada para el intercambio de mensajes entre el código JavaScript de la interfaz y el código Python del servidor. En un principio, se implementó el código sin tener en cuenta que los métodos fueran asíncronos en el servidor por lo que durante las pruebas se producían muchos fallos en la ejecución. La comunicación había sido construida mediante métodos fetch() en el código JavaScript a los que añadías una url como parámetro. Luego en el servidor se serializaban dichos url en cadenas de texto y se comprobaba si dentro de éstos se encontraba una cabecera específica. Según la cabecera analizada se realizaban unas acciones u otras como por ejemplo mover la posición de un brazo o apagar/encender un motor. Aparte de no funcionar como es debido había que solucionar también la comunicación en el sentido contrario, del servidor hacia el JavaScript. Es por eso que buscando otras alternativas se decidió sustituir la implementación por el framework de Microdot el cual incluye una variante para la comunicación utilizando la tecnología websocket. Este protocolo de comunicación permite una conexión persistente y bidireccional de forma asíncrona entre cliente y servidor. Al implementar websockets en nuestro proyecto se elimina el problema de la asincronía de los métodos y la sobrecarga producida por abrir y cerrar conexiones repetidamente. Además, simplifica mucho el desarrollo del código debido a su API ya que resulta mucho más intuitivo y similar por ambas partes de la comunicación. Por tanto, el siguiente paso en el desarrollo del código es crear una conexión websocket que se establezca al comienzo de la ejecución del cliente. También lo óptimo y que hemos realizado en nuestro código es sobreescribir los métodos por defecto del websocket para cuando abrimos la conexión (onOpen), cerramos la conexión (onClose) y recibimos un mensaje (onMessage). Sobreescribirlos

escribiendo un simple mensaje en la consola del navegador ha sido suficiente para así poder saber si se han ejecutado esos métodos o no. Además implementamos el reinicio de la conexión al ejecutar el método Onclose().

```
var url = `ws://${location.host}/ws`;
var websocket;
window.addEventListener("load", onLoad);

function onLoad() {
  iniciarSocket();
}

function iniciarSocket() {
  console.log("Abriendo conexion Websocket...");
  websocket = new WebSocket(url);
  websocket.onopen = onOpen;
  websocket.onclose = onClose;
  websocket.onmessage = onMessage;
}
```

Figura 11. Apertura del websocket

Una vez comentada la tecnología de comunicación vamos a hablar de la funcionalidad de los elementos creados en el HTML. Ya definidas sus respectivas variables en JavaScript les hemos añadido a cada una un “listener” que se activa en cuanto algún componente sea modificado por el usuario. Comenzando por los sliders, sus manejadores de eventos reaccionan en cuanto pulsamos la barra del slider en algún punto o arrastramos la barra, lo que viene a traducirse a cuando se produce un evento “input” en los éstos. Todos sus manejadores actúan del mismo modo, enviando un mensaje en formato JSON hacia el servidor con su valor actual del brazo y su identificador.

Para los manejadores de los botones hemos implementado una funcionalidad muy similar. La diferencia más significativa es que para el mismo elemento hay que desarrollar dos comportamientos diferentes en función de si el motor está activo o apagado en ese momento. Estos botones reaccionan ante un evento de “click” y comprueban si el texto interior es “Apagar”. Si ese es el caso cambia el texto a “Encender” y su color de fondo se convierte en verde. Después envía un mensaje a través del websocket en formato JSON con el valor cero y su identificador de motor. En el caso contrario, si el texto es “Encender” volvería a cambiar el color de fondo a rojo y el texto a “Apagar”. Después enviaría el valor uno y su identificador de motor.

```
botonPower1.addEventListener("click", () => {
  if (botonPower1.innerText === 'Apagar') {
    botonPower1.innerText = 'Encender';
    botonPower1.style.backgroundColor = 'green'
    sendMessage(
      JSON.stringify({
        valor: 0,
        motor: "motor1",
      })
    );
  } else {
    botonPower1.innerText = 'Apagar';
    botonPower1.style.backgroundColor = 'red'
    sendMessage(
      JSON.stringify({
        valor: 1,
        motor: "motor1",
      })
    );
  }
});
```

Figura 12. Ejemplo de manejador del botón de alimentación

Finalmente, para actualizar los datos de lectura en los campos de texto y en la gráfica hemos recurrido a un método periódico que ayuda a una mejor visualización de los valores de fuerza enviados por la máquina de ensayos. Al final del código del JavaScript se declara a la vez que se ejecuta un método llamado “loop” que establece un “setTimeout” (método que ejecuta una instrucción dada al terminar un periodo en milisegundos también dado) de 500 milisegundos que al terminar ejecuta un método que actualiza los lectores y al método “loop”. De esta forma, declaramos un método recursivo que se mantiene activo hasta que se cierra la conexión con el cliente. Dentro del método que actualiza los lectores se realiza una llamada "fetch()" con la url “/updateValues” (el método "fetch()" lo que hace es enviar la url que tiene por parámetro hacia el servidor y busca un método con esa ruta relativa dentro de su dominio actual). Después, lo que devuelva ese método lo hemos convertido en formato JSON y entonces hemos extraído cada valor relacionado con su campo "data" y lo asignamos al campo “value” del objeto referenciado HTML correspondiente a los campos de texto. Asimismo esos valores de retorno se los hemos añadido a los datasets de la gráfica teniendo en cuenta los identificadores de cada brazo para que no se mezclen entre sí. Una vez asignados se llama a la función “update” de la gráfica para que se muestren en la interfaz. Hemos añadido la particularidad de que cuando haya más de diez datos plasmados, cada vez que se añada uno nuevo, el primer elemento se eliminará para que el crecimiento constante de la gráfica no impida al cliente poder ver los datos de forma clara. En conjunto lo que conseguimos es que desde el momento que un cliente se conecte con el servidor, cada 500 milisegundos los campos de texto y la gráfica se actualicen con los datos de lectura provenientes de la máquina hasta que la conexión se vuelva a cerrar.

```
function actualizarLectores() {
  console.log("Recibir señales");
  fetch(`/updateValues`)
      .then((response) => response.json())
      .then(data => {
          lectorBrazo1X.value = data.valor1X;
          lectorBrazo2X.value = data.valor2X;
          lectorBrazo1Y.value = data.valor1Y;
          lectorBrazo2Y.value = data.valor2Y;

          actualizarGrafica([data.valor1X, data.valor2X, data.valor1Y, data.valor2Y]);
      })
}

(function loop() {
    setTimeout(() => {
        actualizarLectores()
        loop();
    }, 500);
})();
```

Figura 13. Función periódica “loop”

5.4. Código del servidor

Esta parte del código corresponde al firmware del microcontrolador escrito en el lenguaje MicroPython que contiene la capa lógica del proyecto y que actúa de intermediario entre la máquina de ensayos y el cliente web. Se ha empezado desarrollando conexiones con el cliente mediante un socket estándar pero debido a complicaciones en la implementación se recurrió a sustituirlas por las conexiones proporcionadas por el framework de Microdot, en nuestro caso websockets.

Hemos empezado por incluir los módulos y bibliotecas necesarias para el desarrollo del código. En éstas hemos incorporado los de Microdot relativos a nuestro proyecto y algunos otros relativos al manejo de componentes, el tiempo y la conexión wifi de la placa. Es más conveniente incorporar tan sólo aquellos módulos de Microdot usemos ya que si ocupamos una gran cantidad de la memoria del microcontrolador esto puede llevar a un peor rendimiento por parte del servidor.

A continuación, hemos declarado las variables referentes a los pines de entrada/salida del microcontrolador. Estudiando la hoja de especificaciones de la placa hemos dividido en grupos de cuatro los pines GPIO para dedicarlos a la alimentación, la dirección y los pasos de los cuatro motores actuadores de la máquina de ensayos. A estos tres grupos habría que añadirle otro más que recoja los datos recibidos por los sensores de la máquina. Sin embargo, ésta al no estar conectada, sus entradas de datos han tenido que ser simuladas por el microcontrolador ya que son necesarias para visualizar los datos en la interfaz. Esto se ha llevado a cabo simplemente mediante un array de datos reales resultantes de un experimento anterior, uno para cada brazo del eje X (dos brazos en total).

Después hemos iniciado la instancia Microdot en la variable “app” y declarado todos los métodos que ésta contiene (los relacionados con el paso de información entre servidor y cliente). Pero antes definimos un método para conectar el microcontrolador a la red wifi con la que vamos a trabajar denominado “conectar”. Éste utiliza el módulo “network” para primeramente crear una instancia WLAN que funciona en modo estación, lo que le permite conectarse a una red wifi ya existente. Luego activa la interfaz de red e intenta conectarse a la red wifi mediante el SSID y la contraseña de la misma, esperando hasta que se conecte para después imprimir en la consola la ip obtenida en su conexión. Esta nos servirá para desde el navegador acceder mediante la ip anterior y el puerto por defecto (en nuestro caso el 5000) a la interfaz web.

```
# Iniciamos aplicacion MicroDot
app = Microdot()
Response.default_content_type = 'text/html'


#Método conexión de placa a red Wifi
def conectar():
    red = network.WLAN(network.STA_IF)
    if not red.isconnected():
        print('Conectando a la red...')
        red.active(True)
        red.connect(SSID, PASSWORD)
        while not red.isconnected():
            pass
    print('Conectado con IP:', red.ifconfig()[0])
```

Figura 14. Creación de la aplicación y el método de conexión

Todos los métodos respectivos a la aplicación Microdot se declaran en relación con una ruta relativa desde la cual el código de la interfaz JavaScript tiene acceso. Esto se indica con la macro “app.route()” a la cual se le pasa como parámetro la ruta relativa a la función que se quiere ejecutar en el servidor, si se accede a dicha ruta mediante el navegador. En primer lugar se define la función “index” en la ruta relativa a la página principal ”/”, a la que el cliente accede simplemente al establecer la conexión con el servidor. Esta función accede a la plantilla HTML creada anteriormente y la renderiza antes de ser devuelta al cliente que realiza la llamada, lo que produce que el usuario lo primero que vea al conectarse con el servicio es la interfaz. Además se define una función muy similar que ofrece en lugar del HTML los archivos estáticos CSS y JavaScript.

En la ruta relativa “/ws” se define el bucle principal de nuestro programa el cúal se mantiene siempre en ejecución atendiendo al envío de mensajes entre servidor y cliente utilizando la conexión websocket. La función asociada a esta ruta se denomina “manejar_aplicacion” y lo primero que realiza es una espera activa (utilizando la palabra clave “await”) a algún mensaje a través del

websocket. Una vez es recibido el mensaje comprueba si en este se encuentra el identificador de un brazo o un motor. Si es el de un brazo ejecuta la función “mover_brazo” y si es el de un motor ejecuta la función “accionar_motor”. A ambas les pasan dos parámetros, su valor y su identificador. A continuación, vamos a explicar el funcionamiento de estas dos.

```
#Iniciamos el websocket
#Aquí implementamos el bucle para recibir valores de clientes web
@app.route('/ws')
@with_websocket
async def manejar_aplicacion(request, ws):

    while True:
        data = await ws.receive()

        if "brazo" in data:

            valor_brazo = ujson.loads(data)
            print("Valor: " + valor_brazo["valor"] + ", Brazo: " + valor_brazo["brazo"])
            mover_brazo(valor_brazo["valor"], valor_brazo["brazo"])

        elif "motor" in data:

            valor_motor = ujson.loads(data)
            print("Power: " + str(valor_motor["valor"]) + ", Motor: " + valor_motor["motor"])
            accionar_motor(int(valor_motor["valor"]), valor_motor["motor"])
```

Figura 15. Bucle principal del código servidor

La función “accionar_motor” comprueba su parámetro relativo al identificador que se le suministra para elegir el motor al que dirigir la acción. Luego se comprueba el parámetro respectivo al valor que puede ser cero o uno (apagar o encender) y se ejecuta el método correspondiente en el pin del motor seleccionado. Por ejemplo, si es el identificador “motor1” y el valor uno entonces se procede a enviar un uno lógico por el pin de salida correspondiente al motor uno con la instrucción “enable_pin1.on()”.

Para la función “mover_brazo” comprobamos de igual forma el parámetro del identificador pero después para calcular el número de pulsos a dar y la dirección a la que ir se requieren más pasos. Primero se ha de crear una variable para almacenar la última posición del brazo (se crea una variable por brazo y se inicializan a cero) y así poder restar el valor enviado como parámetro con ésta. El resultado de la resta es lo que le va a dar la dirección al brazo, porque si el resultado es negativo se moverá hacia valores más pequeños y si es positivo se moverá hacia valores más grandes. De este modo si el identificador es “1x”, su último valor fue cincuenta y le pasamos el valor sesenta su resta será positiva al igual que su dirección. Con la instrucción “dir_pin1.on()” le asignamos al pin de salida correspondiente su valor lógico actual. A continuación, el resultado de la resta anterior se pasa a valor absoluto ya que así obtenemos los pulsos a dar y procedemos a iniciar un bucle con tantas iteraciones como nos indica esta cifra. Para producir estos pulsos mencionados accedemos al pin de salida correspondiente al actual motor para

pasarle un uno lógico, esperamos dos microsegundos, pasamos un cero lógico y volvemos a esperar dos microsegundos. Así cuando estas instrucciones se ejecuten dicho número de iteraciones, el motor habrá producido ese mismo número de pasos.

```
#Método enviar pulsos y direccion a los motores
def mover_brazo(posicion, id_brazo):

    pasos_dados = 0

    if id_brazo == "1x":

        #Calculamos los pasos a dar desde la posicion actual a la siguiente
        num_pasos = int(int(posicion) - int(ult_valor1))

        #Según el resultado anterior decidimos la dirección del motor
        if num_pasos < 0:
            dir_pin1.off()
        else:
            dir_pin1.on()

        num_pasos = abs(num_pasos)

        #Damos tantos pulsos cada 2 us como pasos a dar
        while pasos_dados < num_pasos:
            paso_pin1.on()
            utime.sleep_us(2)
            paso_pin1.off()
            utime.sleep_us(2)
            pasos_dados += 1
```

Figura 16. Método “mover_brazo”

Finalmente hemos implementado la función relativa al muestreo de datos de forma periódica en la interfaz. Como ya hemos comentado anteriormente esta función se relaciona con la ruta relativa “/updateValues” y es accedida desde la interfaz cada quinientos milisegundos. Su función es enviar a través de la respuesta a la petición recibida (la instrucción fetch) una cadena JSON con un diccionario de clave-valor con los valores de la fuerza ejercida por cada brazo de la máquina de ensayos. Estos datos están simulados y se obtienen de a partir de los arrays de experimentos reales que hemos descrito anteriormente. Como tan solo hay datos para los brazos correspondientes al eje X los otros dos siempre obtienen el valor cero.

Una vez implementada toda la funcionalidad necesaria para enviar y recibir datos entre servidor, cliente y máquina de ensayos (si hubiera) lo único que nos quedaba era ejecutar aquellos métodos que hacen que el código funcione correctamente nada más comience a ejecutarse. Para ello creamos una cláusula try-except en la que ponemos en marcha el método de conexión “conectar()” y el método para iniciar la aplicación “app.run()”. La excepción se debe a cuando se produce una interrupción de teclado, en cuyo caso hay que realizar un reset del microcontrolador.

6. Pruebas

Ahora que todo el código está completo, vamos a comprobar que funciona como esperamos. Para ello vamos a ejecutar el código del servidor y del cliente utilizando todas las funcionalidades implementadas. El funcionamiento de la lectura de datos se va a ver claramente mediante la gráfica y los campos de texto, pero las instrucciones enviadas por el cliente usando la interfaz no van a verse reflejadas ya que no hay una máquina de ensayos real a la que conectarse. La manera de sustituir esta prueba es imprimir por consola mensajes de control para saber que el código funciona como se espera por el cliente.

Si ejecutamos el código del servidor podemos ver por pantalla que lo primero que ha hecho la aplicación es conectarse a la red y nos ofrece la ip con la que el cliente se puede conectar desde un navegador.

```
>>> %Run -c $EDITOR_CONTENT

  Conectando a la red...
  Conectado con IP: 192.168.1.135
```

Figura 17. Mensaje por consola de conexión a la red

Al abrir un navegador e introducir la ip y el puerto por defecto (en nuestro caso 5000) podemos comprobar que la aplicación funciona y se puede ver sin problemas la interfaz. Eso prueba que se ha renderizado correctamente la plantilla HTML y el cliente la recibe sin problemas. Además al visualizar así el estilo de los botones también comprobamos que la hoja de estilos CSS se ha cargado junto con el HTML.

Inmediatamente después de abrir la interfaz desde el navegador también podemos observar que la representación de los datos simulados provenientes del servidor actúa como esperábamos, de forma periódica sin que la ejecución falle o se pare en ningún momento. Incluso la característica de que cuando haya 10 puntos representados en la gráfica al mismo tiempo se elimine el más antiguo funciona sin problemas.

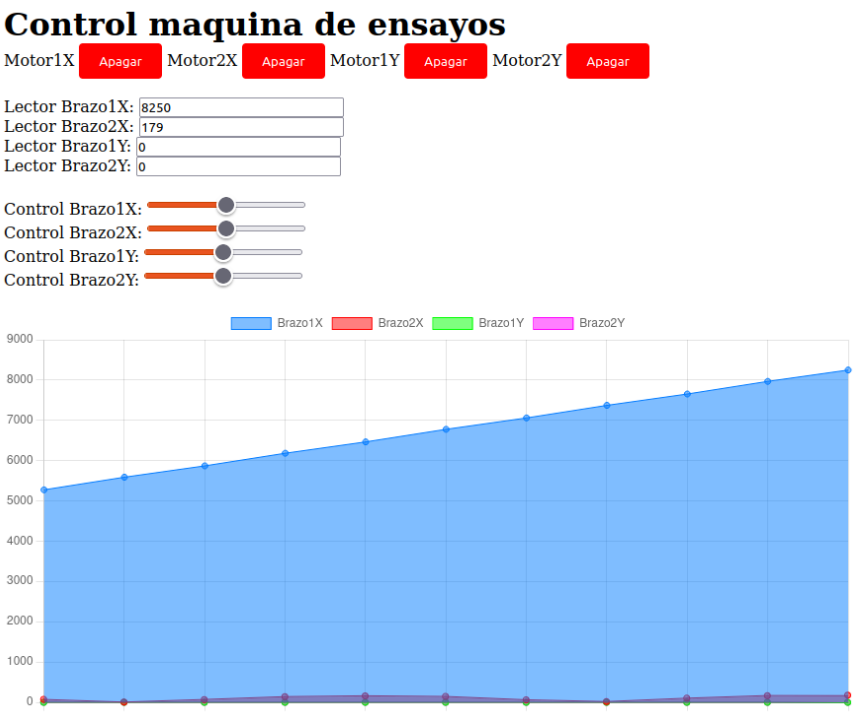

Figura 18. Cliente web en ejecución

Finalmente, hemos comprobado el funcionamiento del JavaScript y el resto del código de control del servidor. Al acceder a la consola del navegador una vez realizada la conexión comprobamos que los métodos del websocket funcionan correctamente.

También el control de los sliders y los botones de alimentación realizan su función correctamente sin alterar los valores aunque la actualización de estos se haga muy rápidamente. Esto demuestra que el diseño asíncrono de los métodos del servidor hacen su función a la perfección.

```
Valor: 22, Brazo: 1x
Valor: 69, Brazo: 1y
Valor: 14, Brazo: 2y
Power: 0, Motor: motor1
Power: 0, Motor: motor3
Power: 1, Motor: motor1
Valor: 70, Brazo: 1y
Valor: 69, Brazo: 1y
Valor: 68, Brazo: 1y
Valor: 67, Brazo: 1y
Valor: 66, Brazo: 1y
Valor: 65, Brazo: 1y
Valor: 64, Brazo: 1y
Valor: 63, Brazo: 1y
Valor: 62, Brazo: 1y
Valor: 61, Brazo: 1y
```

Figura 19. Mensajes por consola del funcionamiento de los manejadores

7. Conclusiones

Como conclusión de este proyecto puedo decir que he cumplido mis objetivos propuestos y que he conseguido ganar un buen conocimiento de un área de la informática de la cuál no tenía ninguna noción, ni de la carrera ni de forma autodidacta. A la vez he podido tener más práctica con nuevas herramientas, lenguajes y tecnologías que me abren la puerta a experimentar con ellas incluso en el uso cotidiano.

Continuando con los objetivos, hemos conseguido completar todos los que nos hemos propuesto empezando por conocer los lenguajes para desarrollar páginas web. Después de terminar el proyecto siento que tengo los conocimientos para poder comenzar a desarrollar páginas web relativamente sencillas incluso no solo destinadas al control de hardware mediante wifi, sino también ligado a bases de datos y dominios en la red. Además el uso del lenguaje Python ha pasado de sentirse algo frustrante, ya que estoy acostumbrado a otros lenguajes estudiados en la carrera como Java o C, a convertirse en otra herramienta con la que me siento cómodo trabajando.

Respecto a la creación de un firmware en la Raspberry Pi Pico W he tenido muchos problemas al empezar. Al ser una placa relativamente nueva y no tan utilizada como su predecesora la Raspberry Pi Pico, me ha resultado bastante complicado encontrar tutoriales, documentación y ejemplos sobre cómo iniciarse en el mundo de IOT (Internet Of Things) utilizando este microcontrolador. Pese a ello, una vez terminado el trabajo puedo decir que aparte de tener un buen conocimiento sobre esta tecnología, he descubierto una gran variedad de proyectos a mi alcance utilizando este tipo de módulos en el día a día.

Todas estas dificultades han conseguido que adquiera una gran capacidad de resolución de problemas complejos que pueden incluso llevar días de investigación solucionar, ya que se convierten en obstáculos que no comprendes cómo comenzar a abordar. Esta clase de esfuerzo no es tan recurrente a lo largo de la carrera y poder practicarlo durante la investigación para el proyecto te prepara para lo que te puede deparar el futuro laboral o académico.

En cuanto a las competencias adquiridas y potenciadas en este proyecto podemos destacar “CT-03 Análisis y resolución de problemas” ya que desarrollar el trabajo desde cero ha implicado afrontar muchos impedimentos. Pero he conseguido afrontarlos de forma modular y sistemática hasta su resolución. También “CT-11 Aprendizaje Permanente” porque el ámbito de este proyecto me ha llevado a aprender técnicas y tecnologías que me van a ser útiles para futuros proyectos. La redacción de éste TFG a su vez ha ayudado mucho a mejorar en “CT-08 Comunicación efectiva” por redactar un texto

complejo estructurando sus ideas y su estilo de forma coherente y estructurada.

En resumen, con este trabajo he conseguido descubrir un nuevo ámbito fascinante de la informática con muchas aplicaciones y un gran recorrido. He conseguido trabajar en dos tecnologías diferentes como son el desarrollo web y el manejo de microcontroladores. Y pese a que no las conocía de antemano he sido capaz de combinarlas para crear este proyecto.

8. Referencias

I. ZwickRoell Materials Testing | ZwickRoell [en línea]. [sin fecha] [consultado el 20 de mayo de 2023]. Disponible en: https://www.zwickroell.com/fileadmin/content/Files/SharePoint/user_upload/PI_EN/09_875_Biax_2kN_horizontally_PI_EN.pdf

II. Startseite von MTS-Systemen [en línea]. Marzo de 2023 [consultado el 20 de mayo de 2023]. Disponible en: https://www.mts.com/-/media/materials/pdfs/brochures/100-335-067_PlanarBiaxial.pdf?as=1

III. TA Instruments [en línea]. 2015 [consultado el 21 de mayo 2023]. Disponible en: https://www.tainstruments.com/wp-content/uploads/sellsheet_TBPlanarBiaxial_022009_low.pdf

IV. eXpert 8000 Series Planar Biaxial Testing [en línea]. 2020 [consultado el 21 de mayo de 2023]. Disponible en: https://www.admet.com/wp-content/uploads/2020/06/eXpert8000PlanarBiaxial_RevA.pdf

V. Microdot — Microdot documentation [en línea]. 2021 [consultado el 20 de junio de 2023]. Disponible en: https://microdot.readthedocs.io/en/latest/index.html

VI. Raspberry Pi Documentation - Raspberry Pi Pico and Pico W. Raspberry Pi [en línea]. 14 de junio de 2023 [consultado el 15 de mayo de 2023]. Disponible en: https://datasheets.raspberrypi.com/picow/pico-w-datasheet.pdf

Anexo 1

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible(ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**				**X**
ODS 2. **Hambre cero.**				**X**
ODS 3. **Salud y bienestar.**	**X**			
ODS 4. **Educación de calidad.**		**X**		
ODS 5. **Igualdad de género.**				**X**
ODS 6. **Agua limpia y saneamiento.**				**X**
ODS 7. **Energía asequible y no contaminante.**				**X**
ODS 8. **Trabajo decente y crecimiento económico.**	**X**			
ODS 9. **Industria, innovación e infraestructuras.**	**X**			
ODS 10. **Reducción de las desigualdades**.			**X**	
ODS 11. **Ciudades y comunidades sostenibles**.				**X**
ODS 12. **Producción y consumo responsables.**				**X**
ODS 13. **Acción por el clima.**				**X**
ODS 14. **Vida submarina.**				**X**
ODS 15. **Vida de ecosistemas terrestres.**				**X**
ODS 16. **Paz, justicia e instituciones sólidas.**				**X**
ODS 17. **Alianzas para lograr objetivos.**				**X**

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados

Una vez terminado el proyecto podemos ver que tiene una fuerte relación con algunos de los Objetivos de Desarrollo Sostenibles en concreto cinco de ellos de los que vamos a hablar a continuación.

Primero vamos a hablar del “ODS 3. Salud y bienestar” ya que está fuertemente ligado a nuestro proyecto. Esto se debe a que las máquinas de ensayos biaxiales se usan principalmente para realizar pruebas sobre materiales y comprobar su comportamiento. Uno de los campos que más utiliza este tipo de pruebas es la biomedicina y la biomecánica que se encargan de testear materiales de prótesis o tejidos orgánicos durante sus experimentos. El hecho de facilitar a estos investigadores este tipo de pruebas hace que las mejoras científicas se vean beneficiadas a su vez.

Por otro lado, el “ODS 4. Educación de calidad” se cumple en nuestro proyecto debido a la naturaleza de código abierto del TFG. Al no restringir el posterior uso del código terminado al ojo público, abrimos la puerta a que muchos estudiantes que quieran explorar este ámbito de la informática puedan tenerlo mucho más fácil. Así pueden inspirarse en él para desarrollar sus propios proyectos, implementar el código en sus programas o incluso mejorar el software para conseguir un resultado final con mejor rendimiento o funcionalidades extra.

Además, podemos afirmar un fuerte compromiso con el “ODS 8. Trabajo decente y crecimiento económico”. Esto se debe a que el objetivo de este proyecto principalmente es ayudar a facilitar un software de control económico de código abierto para usuarios artesanos con pequeñas o medianas empresas. De esta forma, se promueve una digitalización de sus proyectos que les ayudará a mejorar sus técnicas y a aliviar sus cargas de trabajo.

En cuanto al “ODS 9. Industria, innovación e infraestructuras” podemos concluir que hemos aportado una idea innovadora que no se ha visto en la creación de software para máquinas de ensayos a nivel comercial. Ésta sería la capacidad de controlar las máquinas mediante una conexión a Internet desde cualquier dispositivo que pueda acceder a un navegador web. Sin duda, la incorporación de este método de conexión nos acerca más a las técnicas de hoy en día y no implica tener que descargar aplicaciones adicionales innecesarias.

Por último, se puede decir que nuestro proyecto ayuda a cumplir el “ODS 10. Reducción de las desigualdades” ya que como hemos explicado hace dos párrafos nuestro software está diseñado en principio para usuarios artesanos. La gran mayoría de las personas que dirigen una empresa de este estilo no son capaces de permitirse los servicios de creación de estos programas.

Además nunca se les ofrece una opción de compra donde puedan personalizar los parámetros del software en función de su propio hardware a no ser que estén dispuestos a pagar una gran suma de dinero. El avance en proyectos como el nuestro puede solucionar muchos de estos impedimentos económicos y que sus empresas puedan salir adelante.