Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería en Electrónica Industrial y Automática

Diseño Integral e Implementación de un Sistema Mecatrónico Automatizado e Inteligente para la Germinación de Semillas

Autor:

Rodrigo Bratos García

Tutor(es):

Dr. D. Juan Carlos Fraile Marinero

Dr. Dª Ana Cisnal de la Rica

Dpto. Ingeniería de Sistemas y Automática

Valladolid, Marzo 2025.

Agradecimientos

A mis tutores Juan Carlos Fraile Marinero y Ana Cisnal de la Rica, por toda su ayuda y mentoría para la realización de este proyecto.

A mi empresa, ITQ GmbH, por darme la oportunidad de trabajar en este proyecto.

A todos mis compañeros de trabajo, por haberme aceptado e integrado tan rápidamente en la empresa y siempre haber tenido tiempo para ayudarme.

A mis compañeros de proyecto, por todo el trabajo, tiempo y esfuerzo que han dedicado para realizar el proyecto sobre el que trata este TFG.

A Luis Zollbrecht y Tobias Rupert, por haberme proporcionado siempre su ayuda, todos los recursos y herramientas que he necesitado y por haber depositado tanta confianza en mí.

A Jorge Miralles, por haber sido un mentor para mí, siempre guiándome y dándome sus mejores consejos.

A Jorge González, por haber sido tan buen compañero de proyecto desde el primer momento y persona que ha desarrollado y diseñado gran parte del proyecto desde el comienzo.

A todos mis compañeros de clase y profesores durante mis estudios en la Universidad de Valladolid.

A mis amigos y familia por haber estado siempre conmigo y haberme apoyado incondicionalmente.

“I do not know. But I

will try it once more.”

-Hemingway, *The Old Man and the Sea.*

A mis padres.

Resumen y Palabras Clave

Este trabajo presenta el desarrollo de un sistema mecatrónico automatizado para la germinación de semillas, integrando robótica, sensores, actuadores e inteligencia artificial. Se emplea un robot Mitsubishi Melfa RV-7FC-D para la manipulación de macetas y el sembrado, con una Raspberry Pi como núcleo de control. El software, desarrollado en FastAPI y React, permite la gestión del sistema, incluyendo control robótico, monitoreo de sensores, control de actuadores, análisis inteligente de imágenes mediante una API GPT y control por voz con lenguaje natural a través de un modelo de lenguaje de gran tamaño (LLM). Además, se incorpora un Gemelo Digital. El diseño mecánico y eléctrico ha sido optimizado, y la gestión del proyecto sigue la metodología Scrum. Concebido como un demostrador tecnológico, el sistema muestra el potencial de la automatización agrícola y del uso de LLM para el control de sistemas mecatrónicos, fomentando su adopción en la industria.

Palabras clave: Robot Mitsubishi Melfa RV-7FC-D, Raspberry Pi, LLM, control de IO, Gemelo Digital

Abstract and Keywords

This work presents the development of an automated mechatronic system for seed germination, integrating robotics, sensors, actuators, and artificial intelligence. A Mitsubishi Melfa RV-7FC-D robot is used for pot handling and sowing, with a Raspberry Pi serving as the control core. The software, developed using FastAPI and React, enables system management, including robotic control, sensor monitoring, actuator control, intelligent image analysis via a GPT API, and natural language voice control through a large language model (LLM). Additionally, a Digital Twin is incorporated. The mechanical and electrical design has been optimized, and project management follows the Scrum methodology. Conceived as a technological demonstrator, the system showcases the potential of agricultural automation and the use of LLMs for mechatronic system control, promoting their adoption in the industry.

Keywords: Mitsubishi Robot Melfa RV-7FC-D, Raspberry Pi, LLM, IO Control, Digital Twin.

Contenido

Ilustraciones

Tablas

1 Introducción y objetivos

Los contenidos en el presente trabajo de fin de grado, con título “Diseño Integral e Implementación de un Sistema Mecatrónico Automatizado e Inteligente para la Germinación de Semillas” han sido desarrollados durante una estancia de prácticas ERASMUS+ de 6 meses de duración (Septiembre 2024 - Febrero 2025), llevada a cabo en el departamento de ingeniería mecatrónica, específicamente en el proyecto llamado de forma interna “AI_RobotCell”, en la empresa de Ingeniería ITQ (https://www.itq.de/ Marzo, 2025), en la ciudad alemana de Garching bei München, que son las oficinas centrales de la empresa.

Este proyecto se ha realizado bajo la tutoría del Dr. Ing. D. Juan Carlos Fraile Marinero y Ana Cisnal de la Rica profesores del departamento de Ingeniería de Sistemas y Automática de la Universidad de Valladolid, y bajo la supervisión del Msc. -Ing. Tobias Rupert en ITQ.

A continuación, en este capítulo, se procede a describir el contexto en el que se ha realizado el proyecto, lo que incluye una vista general sobre la empresa, un estado del arte tanto de los demostradores como de la automatización de procesos agrícolas y de cuidado de plantas, y posteriormente se presentarán los objetivos de este proyecto.

Debe entenderse que una gran parte del contenido de este documento fue realizado por mí en inglés como documentación interna para la empresa. Esta documentación fue generada en formato Markdown para la “wiki” del proyecto. Dado que he sido el responsable de documentar todo el proyecto y yo soy el autor de todos los contenidos de la “wiki”, he traducido el contenido de esta documentación de forma automatizada utilizando Chat GPT para traducir el contenido y la herramienta Pandoc para transformar el formato de Markdown a Word.

También considero oportuno aclarar que la mayor parte de los contenidos presentes en este trabajo han sido desarrollados por mí, ya sea íntegramente o parcialmente colaborando con otros compañeros, aunque hay algunos campos, como podrían ser el desarrollo del software del frontend y backend o la funcionalidad de reconocimiento de gestos, en las que no he formado parte del desarrollo y simplemente he participado en el diseño, definición de requisitos o la arquitectura, y han sido incluidos en el TFG ya que son fundamentales para una buena comprensión de todo el alcance del proyecto, aunque no se explicaran en gran profundidad.

1.1 La empresa ITQ GmbH

ITQ GmbH es una consultora independiente en ingeniería de sistemas y software con una trayectoria de más de 25 años. Fundada en 1998 por el Dr. Ing. D. Rainer Stetter, ITQ ha logrado posicionarse como una referencia en el sector gracias a su enfoque vanguardista hacia la transformación digital, la integración de sistemas ciberfísicos y el desarrollo de tecnologías innovadoras. Actualmente, la empresa cuenta con cinco sedes, cuatro de ellas en Alemania (Erlangen, Duisburg, Berlín y Garching —donde se encuentra su sede principal—) y una en España, localizada en Las Palmas de Gran Canaria, bajo el nombre de Dr. Stetter ITQ.

En conjunto, la empresa emplea a cerca de 200 personas, destacando por una media de edad menor a los 30 años, lo que refleja su compromiso con la promoción del talento joven.

El modelo de trabajo de ITQ da prioridad a la integración de software en todos los aspectos de sus proyectos, siguiendo la filosofía expresada por el Dr.-Ing. Bernd Spiegelberger, COO de ITQ, en una conversación:

> "En el futuro, ya no habrá 'máquinas con software', sino 'software con máquinas'. Este cambio en el paradigma refleja cómo las máquinas se convierten en un medio para desplegar sistemas digitales complejos."

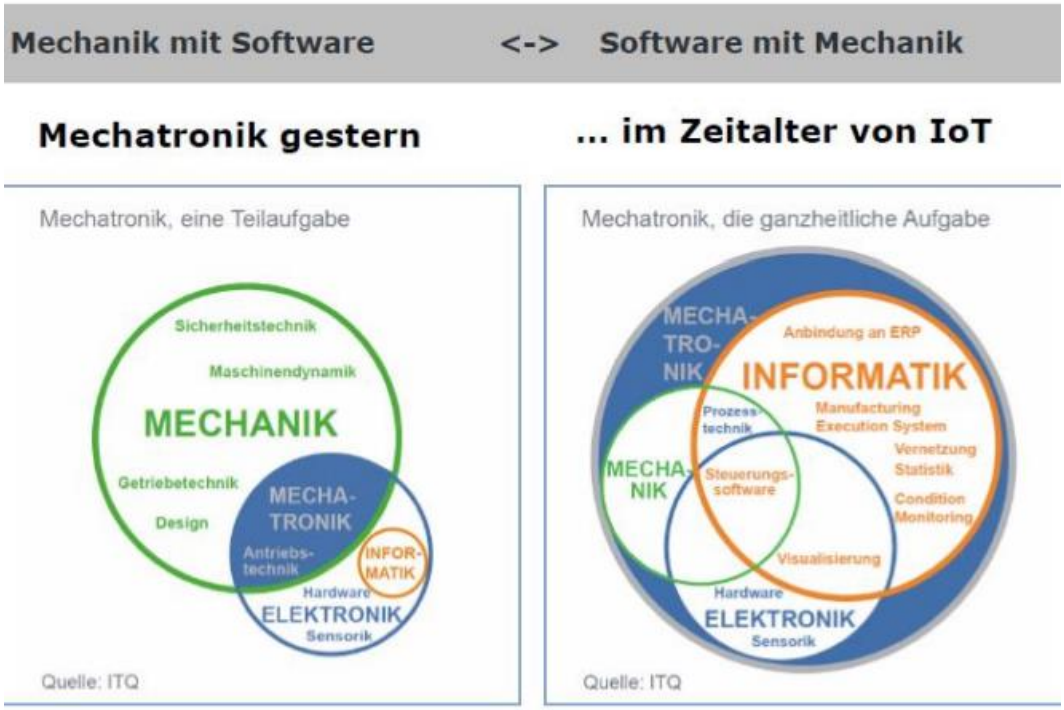

Ilustración 1: Comparación realizada hace 20 años entre la mecatrónica entonces y la predicción del futuro de la mecatrónica según ITQ

1.1.1 Filosofía y Áreas de Enfoque

Desde su fundación, ITQ ha sido un defensor activo de la transformación digital y el avance hacia la Industria 4.0, algo que se puede observar en la Ilustración 1, donde ITQ predijo ya hace 20 años cuál sería la evolución de la ingeniería mecatrónica. La empresa ha sido pionera en proponer que los sistemas de ingeniería deben evolucionar para integrar software desde las primeras etapas de desarrollo. Este enfoque ha influido profundamente en su forma de trabajar y en cómo apoyan a sus clientes en la implementación de métodos de desarrollo modernos. Según el COO:

> "El software no debe simplemente programarse como se hacía en el pasado; ahora es necesario desarrollarlo de forma sistemática. Esto implica adoptar métodos ágiles y eficientes para la ingeniería de sistemas."

Dentro de esta filosofía, ITQ se enfoca en tres pilares principales:

1. **Ingeniería de Sistemas Digitales y Gemelos Digitales**

   ITQ asesora a las empresas para que adopten procesos de desarrollo completamente digitales, empleando tecnologías como los gemelos digitales. Estas herramientas permiten simular y optimizar el comportamiento de sistemas mecánicos y electrónicos antes de su implementación, aumentando la eficiencia y reduciendo costes.

2. **Educación e Impacto Social**

   El compromiso de ITQ con la educación se traduce en iniciativas como su fundación *Technik macht Spaß* (*La tecnología es divertida*), que fomenta el aprendizaje tecnológico desde edades tempranas y promueve proyectos educativos para niños, refugiados y estudiantes. Este enfoque refleja la visión socialmente responsable de la empresa, como señala Bernd:

   > "Ser empresario significa asumir responsabilidad. No solo hacia los empleados, sino también hacia la sociedad. Esto implica operar con rentabilidad, pero sin sacrificar la ética."

3. **Demostradores Tecnológicos para la Industria 4.0**

   ITQ ha dedicado recursos significativos al desarrollo de demostradores tecnológicos como Mi5 y Smart4i, que no solo ilustran las capacidades de las tecnologías modernas, sino que también sirven como plataformas educativas y de inspiración para clientes e industrias. Según el Dr. Spiegelberger:

> “Estos demostradores han sido herramientas clave para motivar a las empresas a adoptar tecnologías innovadoras y metodologías de desarrollo modernas.”

1.1.2 Desarrollo de Demostradores Tecnológicos

El desarrollo de demostradores ha sido una constante desde los inicios de ITQ, cuando la empresa colaboró con equipos estudiantiles para crear prototipos que ilustraran las posibilidades de los sistemas mecatrónicos. En palabras del COO:

> “Estos demostradores no están diseñados para ser productos industriales, sino para mostrar enfoques eficaces y modernos en la ingeniería mecánica y de plantas.”

Impacto de los Demostradores

Los demostradores han tenido un impacto significativo en varios niveles:

- **Educativo:** Proveen un entorno donde estudiantes y jóvenes ingenieros pueden enfrentarse a problemas reales de la industria, aprendiendo a implementar soluciones basadas en tecnologías de vanguardia como inteligencia artificial, robótica avanzada y sistemas ciberfísicos.
- **Inspiracional:** Sirven como ejemplos prácticos para que las empresas entiendan el valor de la transformación digital y adopten nuevas tecnologías.
- **Estrategia Comercial:** Permiten a ITQ mostrar la eficiencia y el valor de sus metodologías de desarrollo, consolidando su posición como líder en el sector.

Ejemplos Clave

Dos de los demostradores más relevantes son los proyectos Mi5 y Smart4i. Ambos integran tecnologías como gemelos digitales y sistemas de inteligencia artificial para simular entornos industriales complejos. Estos demostradores han logrado:

1. Mostrar el potencial de la robótica avanzada en escenarios reales.
2. Impulsar la adopción de métodos modernos en empresas de ingeniería mecánica.
3. Crear sinergias entre tecnología y educación, fortaleciendo la colaboración entre ITQ y sus socios industriales.

1.1.3 Expectativas del Proyecto Actual

El proyecto de germinación automatizada que estás desarrollando representa una continuación natural de la filosofía de ITQ y tiene objetivos específicos alineados con su misión:

1. **Crear un Escenario Educativo Realista:**

   Diseñar un entorno que simule condiciones industriales reales para formar a estudiantes y empleados en el uso de tecnologías avanzadas.

2. **Explorar Límites Tecnológicos:**

   Desarrollar soluciones innovadoras que combinen robótica avanzada, inteligencia artificial y sistemas de visión para llevar la automatización al cuidado de plantas.

3. **Inspirar a la Industria:**

   Mostrar a las empresas del sector mecánico y de plantas cómo integrar soluciones digitales y de inteligencia artificial en sus procesos para avanzar en la transformación digital.

ITQ tiene grandes expectativas para el impacto de este proyecto, esperando que motive tanto a estudiantes como a empresas a explorar nuevas posibilidades. En palabras de su director de operaciones:

> "Este proyecto, al igual que Mi5 y Smart4i, no solo demostrará la eficiencia de las tecnologías modernas, sino que también servirá como un puente entre la academia y la industria, impulsando la digitalización de modelos de negocio."

1.2 Descripción y Motivaciones para el Proyecto

"Diseño integral e implementación de un sistema mecatrónico automatizado e inteligente para la germinación de semillas"

El desarrollo de un sistema mecatrónico automatizado e inteligente para la germinación de semillas aborda los retos actuales de inclusión de LLM e inteligencia artificial para el control de sistemas y de la agricultura moderna: la creciente demanda de alimentos, la presión por el uso eficiente de los recursos y la reducción del impacto ambiental.

Este proyecto integra tecnologías avanzadas de automatización, inteligencia artificial y control ambiental para proporcionar una solución integral y sostenible, a la vez que sirve como un demostrador innovador en ferias de ingeniería, atrayendo interés industrial y fomentando la adopción de nuevas tecnologías.

1.2.1 Descripción del proyecto

En este trabajo de fin de grado se expone cómo se ha desarrollado un módulo automatizado para la plantación, germinación de semillas y cuidado de brotes jóvenes, no solo a nivel de desarrollo e implementación técnica, sino también a nivel de planificación y diseño.

Descripción de funcionamiento

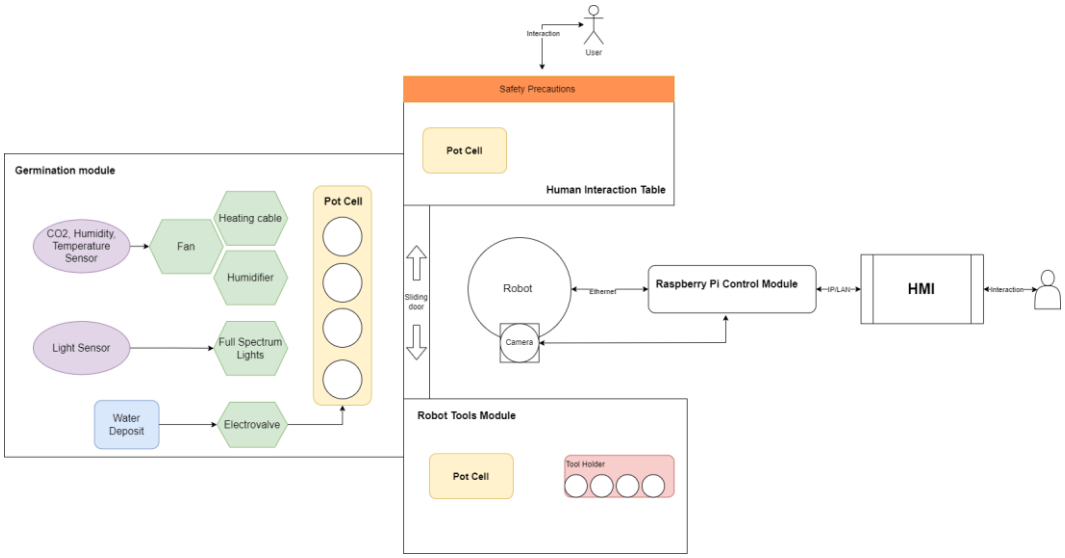

Ilustración 2: Diagrama de Hardware de la célula robótica a vista de pájaro

Esta célula robótica está diseñada para llevar a cabo el proceso de plantado de semillas, y de cuidado de las plantas durante sus primeras semanas de vida de una forma automatizada.

Tal y como se puede observar en el diagrama del hardware del sistema mecatrónico (Ilustración 2, que se encuentra ampliada en el Anejo 4), hay un robot industrial en el centro, que se encargara de realizar diferentes acciones sobre las macetas y plantas. En la parte superior se encuentra la mesa de entrada y salida para las macetas.

En la parte posterior está la cabina con condiciones ambientales aisladas y controladas, donde se encontrarán las plantas mientras germinan y crecen; este módulo consta de una puerta automática que desliza en vertical para permitir el acceso del robot a la cabina. En la parte inferior, inmediatamente al lado del robot, se encuentra el soporte de herramientas, donde el robot acudirá para cambiar su herramienta dependiendo de la acción que deba ejecutar.

Finalmente, a la izquierda del todo de la vista del CAD de la célula robótica (Ilustración 3), se puede ver el cuadro eléctrico, en cuyo interior está la Raspberry Pi y donde se lleva a cabo el control lógico y potencia de todos los elementos eléctricos; en su parte superior se encuentra el controlador del robot.

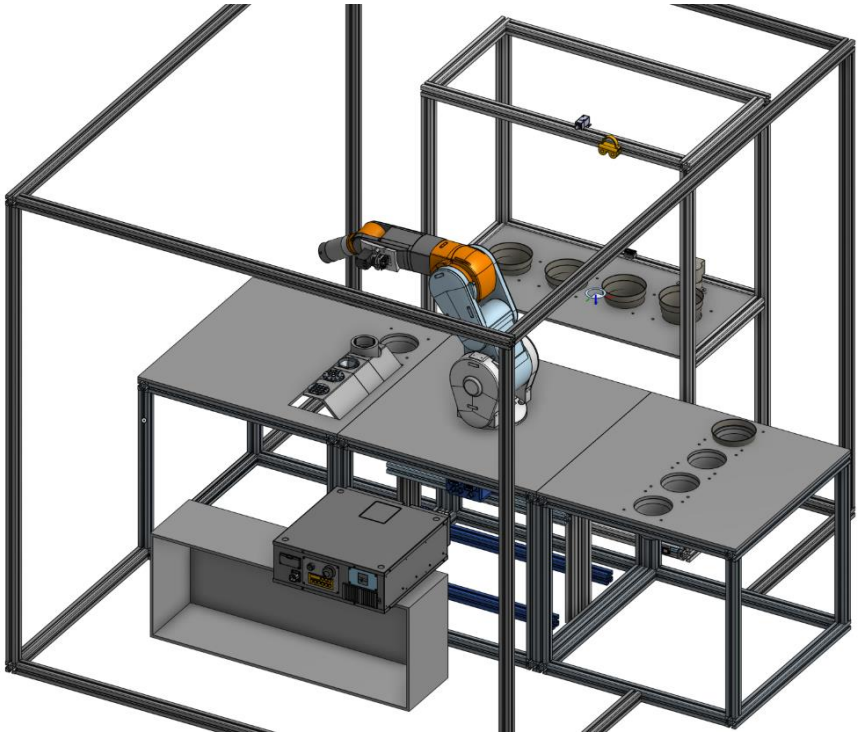

Ilustración 3: Vista general del modelo CAD de la célula robótica para la germinación de semillas

En un ciclo normal de funcionamiento, el usuario introduce una maceta en la mesa de interacción, posteriormente el robot toma dicha maceta con la herramienta especial para la manipulación de éstas, la puerta de la cabina aislada se abre automáticamente permitiendo al robot depositar la maceta dentro del módulo.

Si se da la orden de plantar una semilla, el robot cambia de herramienta y toma la herramienta de sembrado, tras ello, toma una semilla y la introduce dentro de la tierra de la maceta.

Posteriormente, se vuelve a cerrar la puerta automática y el bucle de control para el cuidado de las plantas comienza a controlar las condiciones climáticas dentro de la cabina con los sensores y actuadores; este bucle de control también incluye el regado regular de las macetas, a través de una herramienta especial para el regado, conectada a una manguera con una electroválvula, y hacer que el robot realice una toma de fotos periódica de las macetas, para reconocer su estado de crecimiento a través de inteligencia artificial.

Finalmente, cuando el reconocimiento de imágenes detecta que la planta ha crecido lo suficiente, el robot saca la maceta de la cabina y la deposita en la mesa de interacción.

Además del proceso automático de control, el usuario dispone de un HMI, desde donde puede controlar el módulo de forma manual, con comandos de voz y con gestos: y adicionalmente puede ver toda la información sobre el estado del robot, de los sensores y actuadores y de las plantas.

Descripción de componentes

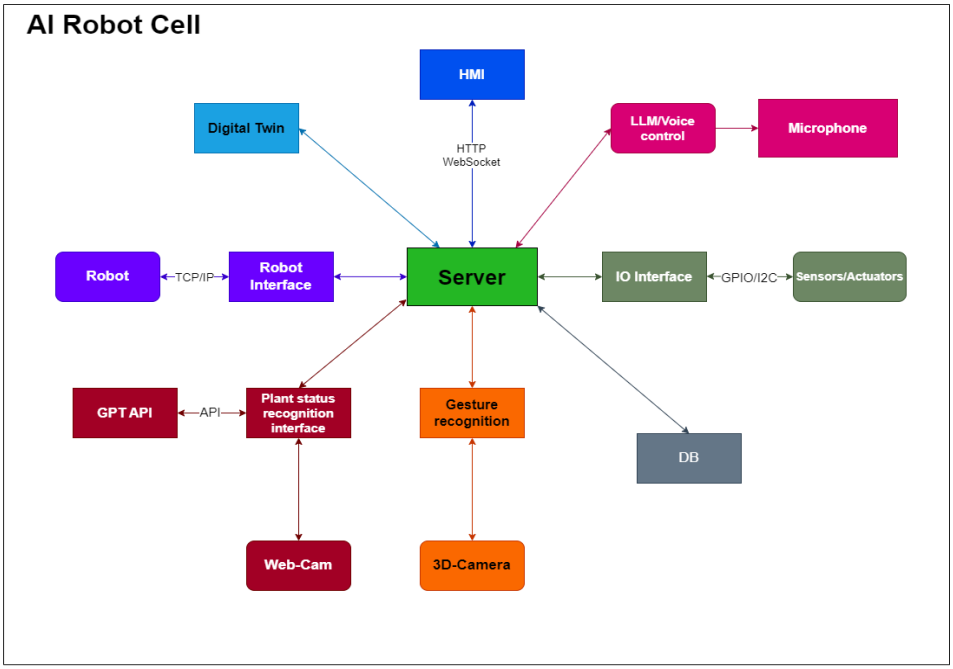

Ilustración 4: Diagrama de alto nivel de la arquitectura del sistema mecatrónico para la germinación de semillas

Para el control de todo el sistema, se ha realizado una Web App o servidor formado por un backend programado en Python con FastAPI y un HMI, o frontend de la Web App, programado con React. Esta aplicación es ejecutada por la Raspberry Pi, que actúa como un PC para controlar todo el sistema, pero que también nos ofrece un módulo de entradas y salidas para poder controlar actuadores y leer sensores.

La Raspberry Pi es el núcleo de control de todo el sistema, que tiene una arquitectura modular, lo que permite la inclusión de nuevas funcionalidades, tal y como se puede observar en la imagen del diagrama de la arquitectura del sistema (Ilustración 4). El backend de la Web App no solo es responsable de controlar el HMI, desde donde se puede visualizar el estado del sistema, realizar acciones manuales o activar los asistentes de inteligencia artificial; sino que también controla el robot Mitsubishi Melfa RV-7FC-D, que es controlado por el servidor a través de una interfaz y un protocolo de comunicación diseñados por nosotros mismos.

Asimismo, la Raspberry Pi controla y se comunica con un gemelo digital, que se ejecuta en un ordenador especial para simulaciones, y que tiene tanto la capacidad de simular los movimientos del robot sin necesidad de disponer de él, como de imitar los movimientos del robot cuando éste se mueve. También controla los sensores y actuadores a través de una interfaz en forma de librería diseñada por nosotros mismos y unos módulos de relés para controlar la potencia de 24 VDC y 230 VAC.

Adicionalmente, el sistema dispone de una interfaz que controla una cámara cuya finalidad es tomar fotos del estado de las macetas o brotes y enviarlas a una API de GPT, que devuelve de forma estructurada el estado de germinación, crecimiento y salud de la planta. También hay un asistente por voz con inteligencia artificial que usa una API de Ollama. A través de este asistente se pueden dar órdenes para plantar, mover macetas, tomar fotos, cambiar el estado de los actuadores o realizar preguntas, ya sean relacionadas con el módulo o no.

Además, actualmente estamos trabajando en incluir un asistente de control por gestos, trabajo que ha sido desarrollado por una compañera como su tesis de fin de máster. Este asistente constará de una funcionalidad a través de la cual se podrán controlar las acciones del robot únicamente por gestos y otra funcionalidad hibrida en la que se podrán utilizar a la vez gestos y voz para controlar el sistema mecatrónico.

Finalmente, hay una base de datos donde se almacenan todos los datos de los sensores y el estado de los actuadores, lo que nos permite representar gráficas en el HMI a través de la herramienta Grafana y tener un mayor conocimiento del estado del módulo para incluir posibles mejoras. Además, en esta base de datos también se guardan todas las fotos de las plantas y las respuestas que la API de GPT devuelve, para así poder representarlas en el HMI y guardarlas, con el fin de, en un futuro, tener un banco de datos lo suficientemente grande y específico como para estudiar la posibilidad de desarrollar un modelo de visión artificial propio utilizando los datos de la base de datos para entrenarlo.

Tareas que se abordan en este TFG

Es oportuno mencionar las tareas que se van a discutir en este trabajo de fin de grado en los próximos capítulos y mencionar brevemente en qué consisten dichas tareas y mi papel en ellas:

1. Modelo CAD (opcional): aquí se describe el diseño de la célula robótica y de los distintos elementos que han sido diseñados y construidos o impresos en 3D, como pueden ser las herramientas del robot o los soportes de los sensores. Debo aclarar que los modelos CAD han sido realizados íntegramente por otros compañeros y mi papel en esta tarea, ya que, como explicaré más adelante en el apartado 2.2, soy el Product Owner del proyecto, ha sido meramente la de la definición de los requisitos de la estación y de los distintos elementos. Aun así, considero que la descripción del diseño de la célula es indispensable para una correcta comprensión del proyecto.

2. Arquitectura del software y de red: en este apartado se describirá cuál es la arquitectura del software de este sistema mecatrónico sin analizar el desarrollo técnico. También se describe la arquitectura de red del

sistema y como se realizan las comunicaciones entre los distintos módulos. En este caso yo solo me he encargado de desarrollar el software de algunos módulos, los cuales se explican en los próximos capítulos, pero no he sido la persona encargada de desarrollar la parte central del software, la Web App, aunque sí que he participado en el diseño de la arquitectura y la definición de los requisitos y sus comunicaciones. Es especialmente importante este capítulo para comprender el funcionamiento del sistema como conjunto y no como módulos aislados.

3. HMI: en este capítulo se discute el diseño de la interfaz humano-máquina del sistema y se realiza una explicación de cómo utilizar y navegar por las distintas ventanas. En el HMI mi aportación ha sido únicamente la de diseñar la experiencia de usuario, conceptualizando y optimizando la interfaz y asegurando que los elementos funcionales y visuales sean accesibles, pero no he formado parte del desarrollo de software correspondiente.

4. Selección de material y diseño eléctrico: en esta parte del proyecto se recogen y explican todos los materiales utilizados, justificando su uso. Adicionalmente se incluirá en este capítulo el diseño eléctrico, lo cual incluye los cálculos eléctricos necesarios, los estándares seguidos, la selección de cableado, diseño de los esquemas eléctricos y del cuadro eléctrico de control. En esta parte del proyecto yo he asumido toda la carga de trabajo.

5. Interfaz IO: esta tarea incluye el análisis de las librerías creadas para el control de los actuadores y la lectura de los sensores desde la Raspberry Pi, la creación de un bucle de control basado en valores limite o gatillo, y una breve explicación del uso de estas librerías para el control de los IO desde el HMI. En este caso yo he sido la persona encargada de crear las librerías de control de los IO y de diseñar el bucle de control automático, además he participado en la inclusión del control de los IO en el HMI, aunque en esa parte he colaborado con otros compañeros para llevarlo a cabo.

6. Robot Mitsubishi Melfa: en este capítulo se expone el robot utilizado, se analizan los programas que realiza el robot, se discute la decisión de haber utilizado una comunicación TCP/IP entre la Raspberry PI y el robot para controlarlo, se enseña un editor y formateador de código Melfa Basic V que se diseñó para poder programar el robot cuando no se disponía de del software de soporte de Mitsubishi y finalmente se explicará en profundidad el protocolo de comunicación diseñado. En cuanto a las tareas asociadas al robot, yo he tomado parte en todas,

aunque no he sido el único desarrollador, habiendo colaborado con otras personas para llevarlo a cabo.

7. Gemelo Digital: en este apartado se muestra como se ha hecho uso del sistema de comunicación diseñado entre el robot y la Raspberry Pi para poder realizar una simulación en tiempo real, especialmente se analiza y explica la configuración del software de simulación para ser controlado por la Web App. He formado parte de esta tarea ayudando a configurar el software de simulación para poder realizar la prueba de concepto, aunque no he sido el desarrollador principal de esta funcionalidad.

8. Análisis automático del estado de las plantas mediante inteligencia artificial: aquí se describe detalladamente como se han utilizado y probado distintos modelos para reconocer el estado de las plantas de una forma automática a partir del análisis de sus imágenes por un agente multimodal (GPT). Se explica en profundidad el proceso de desarrollo y de pruebas realizado hasta llegar a un producto funcional. En este caso yo he sido la persona encargada de desarrollar el código y las pruebas para esta nueva funcionalidad, aunque posteriormente he colaborado con otro compañero para incluirlo en el HMI y backend.

9. Control por voz mediante el uso de un LLM: en este capítulo se explica cómo se ha diseñado el asistente para controlar el sistema mecatrónico con lenguaje natural utilizando un LLM. Se explican las tecnologías utilizadas, la ingeniería de 'prompt' y la lógica de cambio de contexto dinámico utilizando un árbol de decisiones utilizada para obtener una calidad de respuestas óptima. En este caso también he sido yo la persona encargada de diseñar, programar y testear esta funcionalidad, ayudándome de otros compañeros para introducirlo en el sistema.

1.2.2 Motivaciones

Demanda de Eficiencia en la Producción Agrícola

El aumento constante de la población mundial ha incrementado significativamente la demanda de alimentos, lo que obliga a la agricultura a buscar formas más eficientes de producción. La germinación de semillas, al ser la primera etapa crítica del cultivo, requiere una atención especial para garantizar un crecimiento exitoso y uniforme.

Este proyecto utiliza un robot industrial para automatizar y estandarizar procesos manuales complejos y repetitivos de la fase de germinación. La automatización permite:

- Controlar con precisión factores como temperatura, humedad y luz, asegurando un entorno óptimo para la germinación.

- Reducir la variabilidad asociada al trabajo manual y garantizar resultados consistentes en cada ciclo.

El uso de robots industriales en la agricultura ha demostrado aumentar significativamente la productividad y reducir los errores humanos, mejorando la calidad del producto final.

Optimización de Recursos Naturales y Sostenibilidad

La agricultura consume grandes cantidades de agua, energía y nutrientes, lo que representa un desafío ambiental y económico. Este proyecto se centra en la agricultura de precisión, reduciendo el consumo de recursos mediante:

- **Sistemas de riego automatizados** que ajustan el flujo de agua basado en datos en tiempo real de sensores de humedad.
- **Gestión eficiente de iluminación y ventilación**, optimizando el uso de energía mediante tecnología LED y ventilación controlada.

Innovación mediante el uso de Inteligencia Artificial

La incorporación de una cámara para capturar imágenes de las plantas, junto con una API de ChatGPT para análisis instantáneo y la inclusión de un asistente que permite controlar el sistema por comandos de voz, introduce capacidades avanzadas de inteligencia artificial en el sistema:

- **Análisis de imágenes a través de una API GPT:** Detectar enfermedades o deficiencias nutricionales antes de que se conviertan en problemas críticos y sin la necesidad de un experto.
- **Control por voz con el uso de un LLM:** Se pueden realizar acciones en el sistema simplemente dando órdenes con lenguaje natural, sin necesidad de comandos específicos.

La integración de inteligencia artificial en sistemas agrícolas está revolucionando la forma en que se monitorean y gestionan los cultivos, proporcionando una solución personalizada y eficiente para la germinación.

Además, estas funcionalidades podrían ser utilizadas para otras aplicaciones y en muchos otros ámbitos, como en el industrial, en las ciudades o la domótica. Como este proyecto no pretende ser más que un demostrador, estas funcionalidades basadas en LLMs pueden ayudar a fomentar su uso de una forma más generalizada y pueden inspirar nuevos proyectos, ayudando así a superar el escepticismo siempre inherente al sector industrial ante el software y las nuevas tecnologías y a “perder el miedo” a utilizar LLMs para el control de sistemas, ya que en este proyecto se pretende demostrar que con un buen diseño se pueden obtener resultados fiables pese a la falta de determinismo de los LLM.

Reducción de la Intervención Humana y Mejora de la Consistencia

Los sistemas automatizados minimizan la dependencia de la mano de obra humana en tareas repetitivas, como el manejo de macetas y el monitoreo ambiental, reduciendo así los errores asociados. El robot Mitsubishi Melfa ofrece:

- **Alta precisión:** Manipulación de macetas y aplicación de tratamientos específicos basados en datos en tiempo real.
- **Consistencia:** Ejecución de tareas con un nivel uniforme de calidad.
- **Independencia:** El sistema puede funcionar de una forma totalmente independiente sin la necesidad de acciones del agricultor, simplemente notificándole a través del HMI de la información relevante.

Esto no solo libera a los agricultores para realizar actividades de mayor valor añadido, sino que también asegura la calidad del producto final.

Escalabilidad y Adaptabilidad

Desde el comienzo la escalabilidad del proyecto y su adaptabilidad fueron requisitos fundamentales, es por ello por lo que el diseño de la arquitectura del sistema y del software se realizó de forma totalmente modular, ya que no sabíamos si en un futuro podríamos disponer de más personas trabajando en el proyecto, o si tendríamos que cambiar algunos casos de uso del módulo para adaptarlo a la temática de alguna feria de ingeniería concreta. Su arquitectura basada en Raspberry Pi ofrece:

- **Integración con nuevas tecnologías:** Ampliación del sistema con componentes adicionales según sea necesario.
- **Flexibilidad para manejar mayores volúmenes de producción:** Capacidad para adaptarse a las necesidades cambiantes de los productores agrícolas.

Conclusión

El desarrollo de un sistema mecatrónico automatizado para la germinación de semillas está impulsado por la motivación de utilizar y demostrar la viabilidad de nuevas tecnologías, especialmente la inclusión del control de sistemas robóticos con inteligencia artificial y también por la necesidad de mejorar la sostenibilidad y aportar ideas para el desarrollo de soluciones verdes.

Al incorporar robótica avanzada, inteligencia artificial y tecnologías de control ambiental, este proyecto no solo aborda los desafíos actuales del sector, sino que también marca un camino hacia una agricultura más inteligente y sostenible. Como demostrador en ferias de ingeniería, este sistema no solo

fomenta la innovación, sino que también promueve la adopción de nuevas tecnologías en la industria agrícola.

2 Estado del Arte y Metodología para la Gestión y el Desarrollo del Proyecto.

En este capítulo se analizará el estado del arte para este proyecto y además se explicará el proceso de desarrollo del proyecto, así como la metodología de trabajo y mi rol en el proyecto.

2.1 Estado del arte

2.1.1 Estado del Arte sobre Demostradores para Ferias

Introducción

En el ámbito de la ingeniería y la automatización, los demostradores tecnológicos desempeñan un papel estratégico fundamental en ferias y exposiciones. Su objetivo principal es mostrar avances tecnológicos, soluciones innovadoras y aplicaciones específicas de productos o servicios a un público variado que incluye clientes potenciales, socios comerciales, competidores y estudiantes.

Estas herramientas no solo captan la atención y destacan la propuesta de valor de una empresa, sino que también consolidan su posicionamiento en el mercado al demostrar la funcionalidad, eficiencia y beneficios de sus tecnologías en un entorno controlado y visualmente atractivo.

Un demostrador tecnológico es, en esencia, una representación funcional —a escala, simulada o en tiempo real— de un sistema complejo. Su propósito es proporcionar a los asistentes una experiencia inmersiva y tangible que facilite la comprensión de los aspectos técnicos. Esto es especialmente valioso en sectores como la robótica, la mecatrónica y la automatización industrial, donde la sofisticación de los productos puede hacer que las descripciones teóricas o gráficos resulten insuficientes para transmitir su potencial.

Funciones Estratégicas de los Demostradores Tecnológicos

Los demostradores son herramientas versátiles que cumplen múltiples funciones estratégicas, entre las que destacan:

- **Atracción y Captación de Clientes:** Su carácter interactivo permite a los asistentes experimentar directamente los beneficios del producto o tecnología, incrementando la probabilidad de generación de leads, ventas y alianzas estratégicas.
- **Demostración de Liderazgo Tecnológico:** Empresas que desarrollan demostradores innovadores se posicionan como referentes de vanguardia en su sector. Esto fortalece su marca y genera confianza en sus capacidades.

- **Validación Tecnológica:** Al presentar una simulación funcional de tecnologías en un entorno controlado, las empresas pueden validar el desempeño de sus soluciones frente a las demandas reales del mercado.
- **Educación y Formación:** Los demostradores son también una herramienta educativa fundamental. Estudiantes y jóvenes ingenieros suelen participar en su diseño y construcción, adquiriendo experiencia práctica en la resolución de problemas técnicos y en el uso de tecnologías avanzadas. Este enfoque no solo fomenta el aprendizaje, sino que también ayuda a las empresas a identificar y formar talento.

Ejemplos Destacados de Demostradores de ITQ

A continuación, se analizan tres casos de demostradores recientes realizados en ITQ **[1]** que destacan por su innovación, impacto educativo y capacidad de generar interés:

1. Google Cloud Demonstrator: Entendiendo Google Cloud – ITQ

Este demostrador (Ilustración 5), desarrollado en colaboración entre ingenieros jóvenes de ITQ GmbH y SOTEC, ejemplifica cómo las soluciones de Google Cloud pueden revolucionar los procesos productivos mediante tecnologías como inteligencia artificial y análisis avanzado de datos.

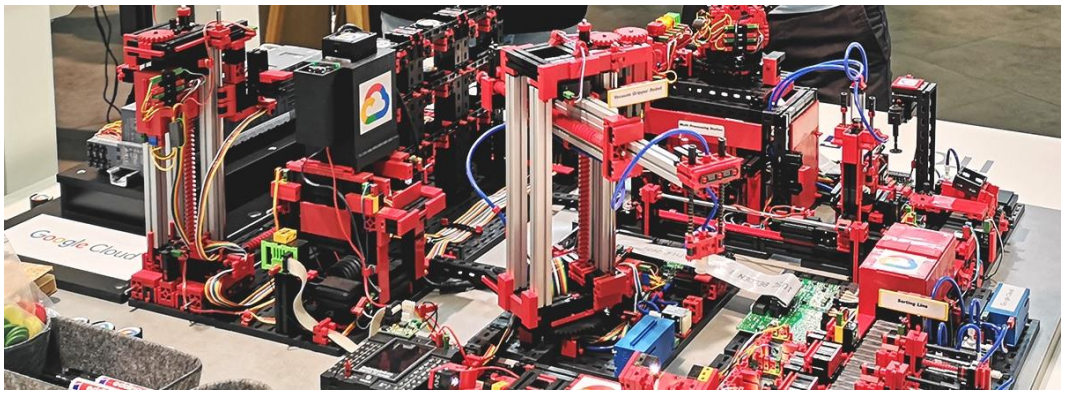

Ilustración 5: Demostrador Google Cloud hecho en ITQ

Características destacadas:

- Representa una minifábrica que produce chips personalizados.
- Utiliza el *Manufacturing Data Engine* y *Connect* de Google Cloud para optimizar procesos mediante análisis predictivo, mantenimiento preventivo, inspección visual y detección de anomalías.
- Emplea un robot colaborativo Sawyer de Rethink Robotics, que opera en tiempo real para ejecutar tareas específicas de producción.

Este demostrador no solo es una herramienta educativa, sino que también ilustra el potencial de las fábricas inteligentes habilitadas por soluciones en la

nube, marcando un referente para el uso de inteligencia artificial y robótica en entornos productivos.

2. Fwip Ice Demonstrator: La Máquina de Helado Inteligente

Este proyecto combina la robótica colaborativa con una máquina de helados de cápsulas de la marca Fwip (Ilustración 6). Diseñado por estudiantes, este demostrador automatiza todo el proceso de preparación y entrega de helados.

Ilustración 6: Demostrador Fwip Ice hecho en ITQ

Características destacadas:

- Integración de motores, sensores y un robot Sawyer que selecciona el vaso, prepara el helado y lo entrega al usuario.
- Uso de luces LED para añadir atractivo visual y captar la atención del público.
- Aplicación de principios de IoT para coordinar las diferentes partes del proceso.

Este demostrador es un ejemplo notable de cómo la robótica puede integrarse en productos de consumo para optimizar procesos cotidianos, mostrando aplicaciones prácticas del IoT y la automatización.

3. Mi5 Dartboard: Diana de Alta Velocidad

El Mi5 Dartboard representa una combinación impresionante de control predictivo, robótica avanzada y atractivo visual (Ilustración 7). Diseñado para transformar una actividad recreativa común, este demostrador asegura que cada dardo impacte en el centro de la diana. Este rpoyecto fue pionero en la inclusión de visión artificial y control predictivo.

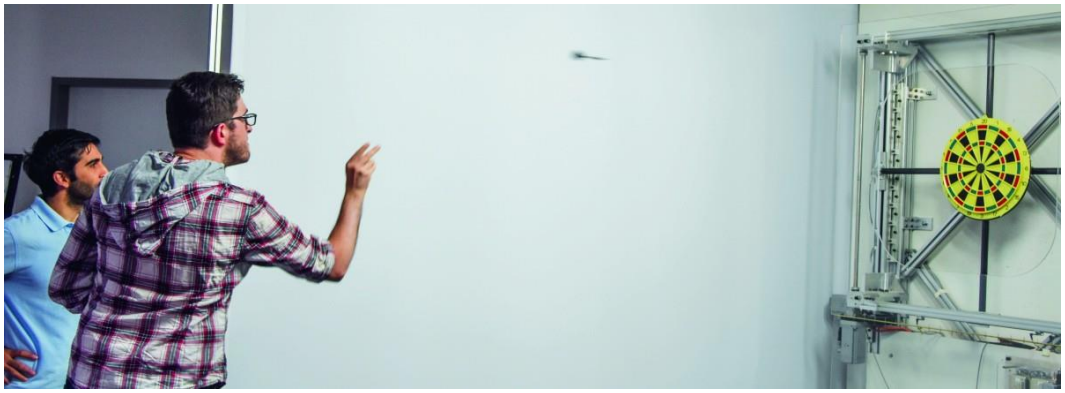

Ilustración 7: Demostrador Mi5 Dartboard hecho en ITQ

Características destacadas:

- Utiliza cámaras de alta velocidad para rastrear la trayectoria del dardo en tiempo real.
- Implementa algoritmos de predicción para calcular el punto de impacto y mover la diana a velocidades extremadamente altas (hasta 12g de aceleración y 30g en frenado).
- Proporciona una experiencia única para el público, destacándose como una atracción interactiva en ferias.

Además de su función lúdica, este demostrador ejemplifica la precisión y capacidades de los sistemas de control predictivo en la ingeniería avanzada.

Tendencias en el Desarrollo de Demostradores

El estado del arte en demostradores tecnológicos está definido por varias tendencias clave:

1. **Integración de Inteligencia Artificial (IA):** Cada vez más demostradores incluyen tecnologías de IA para realizar análisis predictivos, tomar decisiones autónomas y optimizar procesos en tiempo real.
2. **Uso de Gemelos Digitales:** Herramientas que permiten simular el comportamiento de un sistema físico en un entorno virtual, facilitando la validación y optimización de diseños antes de su implementación.
3. **Enfoque en Sostenibilidad:** Muchos demostradores ahora incluyen tecnologías diseñadas para minimizar el impacto ambiental, como la eficiencia energética y el reciclaje de materiales.
4. **Interactividad con el Usuario:** Los demostradores actuales buscan experiencias altamente inmersivas e interactivas para captar la atención del público y fomentar el aprendizaje práctico.

Conclusión

Los demostradores tecnológicos son herramientas esenciales para la promoción y validación de avances tecnológicos en ferias y exposiciones. A

través de ejemplos como el Google Cloud Demonstrator, el Fwip Ice Demonstrator y el Mi5 Dartboard, queda claro que estas iniciativas no solo benefician a las empresas al mostrar su liderazgo tecnológico, sino que también desempeñan un papel crucial en la formación de las nuevas generaciones de ingenieros.

Los demostradores no solo permiten visualizar el futuro de la automatización y la robótica, sino que también actúan como catalizadores para la innovación y la adopción de tecnologías complejas en diversos sectores.

2.1.2 Estado del Arte sobre Módulos de cuidado de plantas

El avance en la mecatrónica y la inteligencia artificial ha permitido el desarrollo de sistemas automatizados para la agricultura de precisión. En particular, la combinación de robótica, sensores y algoritmos de inteligencia artificial ha optimizado procesos clave como la siembra, monitoreo y cuidado de cultivos en sus primeras etapas de vida.

En este contexto, se han desarrollado múltiples enfoques en la automatización de la germinación de semillas, desde robots sembradores hasta sistemas autónomos de monitoreo y cuidado de cultivos.

Este estado del arte analiza las investigaciones recientes en este campo para sentar las bases del diseño de una célula robótica automatizada para la siembra y germinación de semillas.

Aplicación de Inteligencia Artificial, Robótica y Sensores en la Agricultura

Uno de los avances más significativos en la agricultura ha sido la integración de inteligencia artificial (IA), sensores y robótica en la fenotipificación de plantas y la agricultura de precisión. Estos sistemas permiten monitorear en tiempo real variables como la humedad del suelo, las condiciones climáticas y el estado de salud de los cultivos, facilitando la toma de decisiones automatizadas y reduciendo la intervención manual [2].

Adicionalmente, en el anteriormente citado artículo, también se menciona como se realizó un modelo de visión artificial para la detección temprana de enfermedades en las plantas.

En relación con el sistema mecatrónico propuesto en esta tesis, estas tecnologías permiten la implementación de un control automatizado basado en sensores para la regulación de las condiciones climáticas dentro de la cabina de germinación.

También se incluye el uso de visión artificial para evaluar el estado de crecimiento de las plantas, así como su estado de salud, aunque, a diferencia del articulo citado, en este proyecto se ha utilizado y adaptado un modelo multimodal de propósito general (GPT API) para llevar a cabo la fenotipificación

de las plantas y reconocer su estado de germinación, crecimiento y salud; pero a diferencia del caso expuesto en el artículo [2], sin tener que diseñar y entrenar un modelo específico.

Automatización de la Siembra y Germinación de Semillas

La automatización de la siembra de semillas ha sido ampliamente estudiada en la literatura. Un estudio relevante presenta el desarrollo de un robot de siembra inteligente con un brazo mecánico que permite la plantación automatizada de semillas mediante un sistema de succión de aire [3]. Este robot, controlado a través de una aplicación móvil, optimiza la distribución de semillas y minimiza la intervención humana en el proceso de siembra.

Otro enfoque importante es la mejora de sistemas de siembra mediante la optimización de rutas y la capacidad de almacenamiento de semillas [4], en el cual se utilizan y mejoran las herramientas del robot Farmbot Genesis [5], un proyecto que es de código abierto, incluyendo también los modelos CAD de las herramientas.

Estos desarrollos son relevantes para el proyecto sobre el que trata este trabajo de fin de grado, ya que la célula robótica diseñada en este trabajo debe incorporar un mecanismo de plantado eficiente, para lo cual se han tomado las herramientas del Farmbot Genesis y han sido rediseñadas y adaptadas al robot industrial Mitsubishi Melfa.

Uso de LLM y Reconocimiento de Gestos en el Control de Robots

El uso de **Modelos de Lenguaje de Gran Escala (LLM)** en la robótica ha permitido interpretar comandos en lenguaje natural, facilitando la interacción humano-robot sin necesidad de programación explícita [6]. Estos modelos ofrecen flexibilidad y adaptabilidad en la ejecución de tareas, pero presentan desafíos en la interpretación de comandos ambiguos y la integración con otros sistemas sensoriales [7].

Por otro lado, el control por gestos se ha desarrollado como una alternativa intuitiva para la manipulación robótica, permitiendo la interacción sin contacto y aumentando la accesibilidad del sistema [8]. La combinación de comandos por voz con LLM y reconocimiento de gestos mejora la precisión y eficiencia en el control del robot, alineándose con las tendencias actuales en interfaces naturales para robótica.

Conclusiones del estado del arte

El estado del arte revisado demuestra que la integración de inteligencia artificial, robótica y sensores en la agricultura ha permitido desarrollar sistemas altamente automatizados para la siembra, monitoreo y cuidado de cultivos. La

presente investigación se basa en estos avances para diseñar un sistema mecatrónico automatizado de germinación de semillas que incorpora:

- Un robot industrial para manipulación de macetas y siembra de semillas.
- Sensores y actuadores para el control de condiciones ambientales dentro de la cabina de germinación.
- Un sistema de visión artificial para evaluar el crecimiento de las plantas.
- Un gemelo digital para la simulación y optimización del proceso de pruebas.
- Un HMI avanzado con control por voz y gestos.

Estos elementos, en conjunto, permitirán mejorar la eficiencia del proceso de germinación de semillas y reducir la necesidad de intervención manual, alineándose con las tendencias actuales en automatización agrícola.

2.2 Gestión del proyecto y metodología de trabajo.

Durante el desarrollo del proyecto la planificación y el método de gestión han sido fundamentales para poder lograr los objetivos. Es especialmente importante mencionar la metodología de organización y de trabajo del proyecto y como se ha diseñado esta, ya que yo asumí el rol de 'Product Owner' del proyecto (Metodología Scrum) y me encargué, en parte, de diseñar dicho método de trabajo.

2.2.1 Etapas del proyecto

Desde mi punto de vista, durante el desarrollo del proyecto ha habido dos etapas bastante diferenciadas en cuanto a la metodología del trabajo y la forma de gestionar el proyecto. Aunque en cuanto al desarrollo o a un nivel interno de la empresa no se ha diferenciado oficialmente en etapas el proyecto, sí que considero importante exponer el contexto de la metodología que hemos seguido a lo largo de los seis meses de trabajo.

Primera Etapa de Organización

En los primeros meses de trabajo en el proyecto, a nivel interno llamado AI_RobotCell, existían varias líneas de desarrollo en paralelo, unos compañeros estaban centrados en un robot Comau controlado por un autómata B&R, otros compañeros estaban centrados en la investigación para controlar robots a través de reconocimiento de gestos con una cámara 3D, y mi compañero Jorge González y yo comenzamos a trabajar con el Robot Mitsubishi Melfa RV-7FC-D.

A comienzos de nuestra estancia el objetivo era conseguir que se moviera alguno de los robots, ya que el robot Comau no se movía debido a problemas

con licencias y el robot Mitsubishi tampoco, ya que no se disponía del software oficial de Mitsubishi para programarlo y simularlo.

Durante esta primera fase del proyecto, un empleado de ITQ, Jorge Miralles, que hizo de mentor para nosotros, nos introdujo a la metodología Scrum (explicada en el apartado 2.2), en la que mi compañero y yo éramos los desarrolladores y él se encargaba de los roles de Product Owner y Scrum Máster. En este periodo el método Scrum fue introducido con un objetivo más bien didáctico, para familiarizarnos con él, pero con el paso del tiempo pudimos observar que nos daba muy buenos resultados, ayudándonos a desarrollar el proyecto de una forma rápida y organizada.

En estos primeros tres meses trabajando mi compañero y yo como desarrolladores conseguimos diseñar nuestro propio método de comunicación con el controlador del robot utilizando TCP/IP y evitando así la necesidad de adquirir el software de Mitsubishi y un PLC para controlar el robot. Aprovechando este método de comunicación, decidimos utilizar una Raspberry Pi 4B para controlar todo el sistema y diseñamos nuestro propio Backend, Frontend o HMI, el control de los sensores y actuadores.

También durante esta etapa configuramos un Gemelo Digital utilizando ISG-Virtous, realizamos un sistema de reconocimiento del estado de las plantas utilizando una API de GPT y el Proof of Concept (prueba de concepto, PoC) de un asistente por voz basado en un LLM para controlar el sistema.

Segunda Etapa de Organización

Una vez transcurrieron dos meses, tuvimos una reunión con el resto de las personas del proyecto y nuestros supervisores para intentar que todas las personas trabajásemos juntas para un mismo objetivo en vez de trabajar en paralelo. Tras esta reunión se decidió que finalmente utilizaríamos el robot Mitsubishi como único robot del proyecto.

Para esta fase establecimos como objetivos el diseño y construcción de la estructura, herramientas y controles eléctricos, la integración del control por voz con IA y del control por gestos y la mejora del gemelo digital para que el robot interactúe con la estación en las simulaciones. Para este periodo se estableció que trabajaríamos con metodología Scrum como un solo equipo, en el que a mí se me asignó el rol de Product Owner.

2.2.2 Método Scrum

La metodología Scrum [9] fue utilizada como marco de trabajo ágil para el desarrollo de nuestro proyecto. Scrum es un enfoque iterativo e incremental diseñado para gestionar proyectos complejos, promoviendo la adaptabilidad y la generación de valor continuo.

Fundamentos de Scrum

Scrum se basa en dos principios clave: **empirismo** y **pensamiento Lean**. El empirismo destaca que las decisiones deben basarse en la experiencia y la observación directa, mientras que el pensamiento Lean busca minimizar desperdicios y concentrarse en lo esencial. Este marco emplea ciclos de trabajo llamados *Sprints*, que son periodos de tiempo predefinidos (normalmente de 2 a 4 semanas), en los que el equipo trabaja para entregar un incremento del producto con valor añadido.

Además de dividir el proyecto en estos periodos temporales también se divide el trabajo en 'Issues' o tareas, siendo la duración de estas tareas de como máximo un sprint, aunque lo ideal es que un desarrollador termine varias tareas por sprint, por lo tanto, las tareas demasiado complejas o duraderas deberán desglosarse en tareas más cortas y simples.

Scrum fomenta un conjunto de valores fundamentales: **Compromiso, Enfoque, Apertura, Respeto y Coraje**, que guían las interacciones y decisiones del equipo para asegurar un entorno colaborativo y eficiente.

Roles en Scrum

El equipo Scrum está compuesto por tres roles principales:

Product Owner (Propietario del Producto)

El **Product Owner** es responsable de maximizar el valor del producto y garantizar que el equipo desarrolle lo que es más importante. Sus responsabilidades clave incluyen:

- Definir y gestionar el **Product Backlog** (una lista priorizada de tareas y características).
- Asegurar que el equipo entienda los requisitos y los objetivos del negocio.
- Comunicarse con las partes interesadas y recopilar retroalimentación.
- Tomar decisiones sobre la prioridad del trabajo en función de las necesidades y el valor del proyecto.

Enfoque Principal: Visión, priorización y comunicación.

Para llevar a cabo mi rol de Product Owner seleccione unos métodos de priorización de tareas y de comunicación. Posteriormente genere una documentación (apartado 2.4) para ser transparente con mi equipo.

Scrum Máster

El **Scrum Máster** es un facilitador y líder servidor que asegura que el equipo siga el proceso Scrum. Sus responsabilidades incluyen:

- Eliminar obstáculos que dificulten el progreso del equipo.
- Facilitar eventos Scrum, como el Daily Standup, Sprint Planning, Sprint Review y Sprint Retrospective.
- Entrenar al equipo en principios y prácticas ágiles.
- Asegurar un entorno de equipo saludable y colaborativo.

Enfoque Principal: Facilitación de procesos, soporte al equipo y mejora continua.

Desarrollador

Los **Desarrolladores** son los miembros del equipo que trabajan colaborativamente para entregar incrementos del producto. Sus responsabilidades incluyen:

- Descomponer tareas y entregar trabajo incremental durante cada sprint.
- Participar en Sprint Planning, Daily Standups y otras ceremonias Scrum.
- Asegurar que la calidad del trabajo cumpla con la Definición de Hecho (DoD).
- Colaborar con el Product Owner y el Scrum Máster para cumplir los objetivos del sprint.

Enfoque Principal: Construcción, prueba y entrega de incrementos funcionales del producto.

En nuestro caso, tanto el PO como el SM también fueron desarrolladores.

Tabla Resumen

En la Tabla 1 puede verse un resumen de los roles del Scrum.

Tabla 1: Tabla resumen de los roles del Scrum

Rol	Responsabilidades Clave
Product Owner	Definir prioridades, gestionar el backlog y planificar el futuro del proyecto.
Scrum Máster	Facilitar el proceso, eliminar obstáculos y apoyar al equipo.
Desarrollador	Entregar incrementos de producto de alta calidad, colaborar y cumplir objetivos.

Artefactos de Scrum

Scrum utiliza tres artefactos fundamentales que fomentan la transparencia y el control del progreso:

1. **Product Backlog:** Lista priorizada de requisitos y tareas del producto.

2. **Sprint Backlog:** Conjunto de elementos seleccionados del *Product Backlog* para completarse durante el Sprint.
3. **Incremento:** Resultado tangible y utilizable al final de cada Sprint.

Cada artefacto incluye un compromiso asociado que asegura su alineación con los valores y objetivos del equipo: - Meta del Producto, Meta del Sprint, y la Definición de Hecho (*Definition of Done*).

Eventos Clave

Los eventos estructurados en Scrum aseguran ciclos regulares de inspección y adaptación:

- **Sprint Planning:** Planificación del Sprint, estableciendo la Meta del Sprint y seleccionando las tareas a realizar.
- **Daily Scrum:** Reuniones diarias de 15 minutos para inspeccionar el progreso y ajustar el trabajo.
- **Sprint Review:** Evaluación de los resultados del Sprint con los interesados para definir los próximos pasos.
- **Sprint Retrospective:** Reflexión del equipo sobre lo que funcionó y qué mejorar.

Diagrama del Método Scrum

En la Ilustración 8 se muestra el flujo de trabajo del método Scrum.

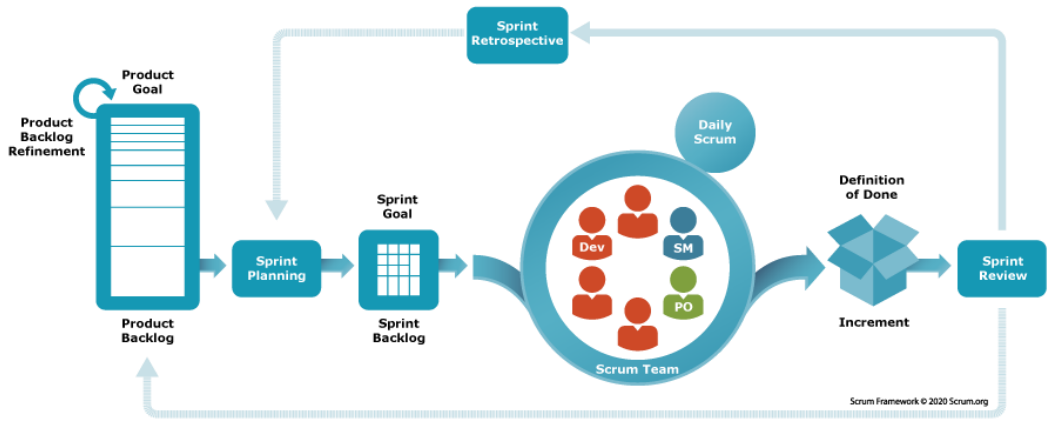

Ilustración 8: Flujo de trabajo del método Scrum

2.2.3 Flujo de Trabajo y Método de Trabajo del Proyecto Usando Scrum

En este apartado se explica cómo se ha implementado el método Scrum y como ha sido el flujo de trabajo del equipo.

Issue Board

En este proyecto trabajamos con un 'Issue Board' (Ilustración 9) o tablero de tareas, del que yo, como Product Owner, he sido responsable de diseñar y mantener:

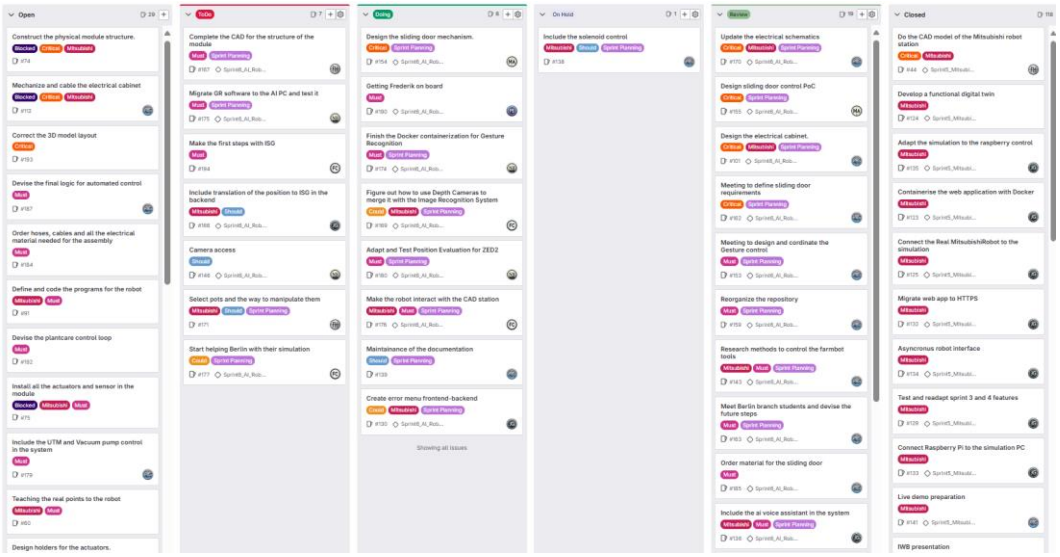

Ilustración 9: Issue Board del proyecto

Nota: en el Anejo 4 se podrá ver ampliado el tablero de tareas.

Dentro de este tablero hay 6 columnas diferentes:

- Open (no programadas para el sprint actual)
- ToDo (programadas para el sprint actual pero no iniciadas)
- Doing (programadas e iniciadas durante el sprint actual)
- OnHold (iniciadas, pero bloqueadas por algo o alguien)
- Review (completadas y esperando revisión en la próxima revisión del sprint)
- Done (revisadas y aceptadas)

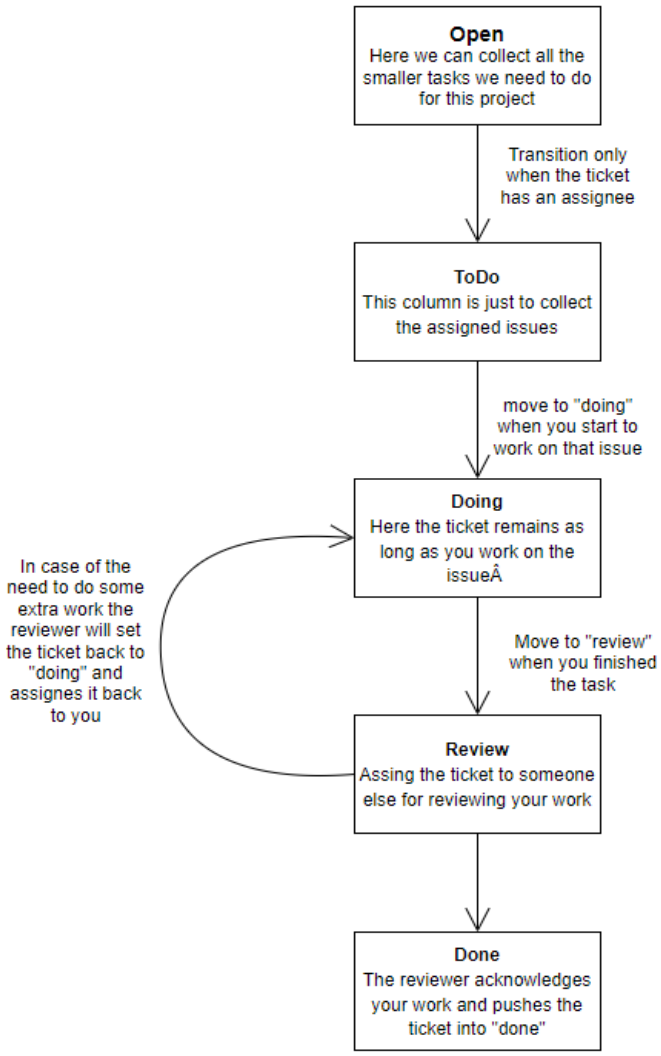

Ilustración 10: Flujo de las tareas para nuestro Issue Board

Tal y como se puede observar en la Ilustración 10, cada vez que una tarea cambia de estado, deberá ser arrastrada a la siguiente columna. De esta manera el Product Owner puede realizar un seguimiento del trabajo durante el sprint y obtener una idea del estado actual del desarrollo.

Código de Etiquetas

- **Prioridad**: se enumeran las etiquetas de prioridad de más importante a menos importante:
    - Critical: la prioridad más alta para las tareas; una característica esencial que bloquea o es necesaria para comenzar otras características esenciales.
    - Must: una característica que el proyecto debe tener al final del sprint.

    - Should: una característica que el proyecto debería tener al final del sprint.
    - Could: una característica que sería agradable tener al final del sprint.
    - Won't: una característica que el proyecto no tendrá por ahora.
- **Sprint planning**: Esta etiqueta se usa para diferenciar las tareas planificadas al inicio del sprint de las que se volvieron necesarias durante el sprint. Esto nos permitirá evaluar la calidad de la planificación del sprint y mejorar en los siguientes.

Reuniones del Equipo

Reuniones diarias de Scrum (Stand-Up)

- Frecuencia: dos veces por semana (miércoles a las 9:30 y viernes a las 9:30).
- Tema: ¿en qué estás trabajando actualmente? ¿Tienes algún problema?
- Duración: 15 minutos.

Día de Oficina

- Día: miércoles.
- Explicación: día en que todos vienen a la oficina para trabajar juntos y ayudarse mutuamente.

Revisión del Sprint y Planificación del Sprint

- Frecuencia: cada 3-4 semanas.
- Duración: aproximadamente 2 horas.
- Tema: revisar el trabajo de las semanas anteriores y recibir retroalimentación. Adicionalmente, utilizaremos esta reunión para planificar el próximo sprint y definir sus objetivos.

Información del Equipo

En la Tabla 2 se pueden ver los participantes del proyecto y sus roles.

Tabla 2: Tabla de la información del equipo

Nombre	Rol	Tareas	Horas de trabajo semanales
Nicola	desarrolladora	Software, IA	10-16h semanales
Finn	desarrollador	CAD, diseño mecánico	10h semanales
Jorge	Scrum Máster, desarrollador	SM, software, mecatrónica, ingeniería de sistemas	40h
Rodrigo	Product Owner, desarrollador	PO, IA, mecatrónica, ingeniería de sistemas	40h
Fabrice	desarrollador	Gemelo Digital	40h
Ali	desarrollador	Ingeniería mecatrónica	20h
Frederik	desarrollador	Ingeniería de software	15-20h
Simla	desarrolladora	Aprendizaje por refuerzo, software	16h (Berlín)
Sandra	soporte técnico	Definición de requisitos, ingeniería de sistemas	NA
Luis	supervisor	Supervisión, soporte técnico	NA

2.2.4 Metodología de trabajo como Product Owner

Como Product Owner, mi responsabilidad principal es asegurar que el Equipo Scrum entregue el máximo valor a final de cada sprint y, especialmente al final del proyecto. Para lograr esto, me comprometí a:

- Priorizar el backlog de manera efectiva utilizando el método **MoSCoW**.
- Comunicarme de manera clara y colaborativa aplicando los criterios **INVEST** para las historias de usuario dentro de la definición de las tareas en el Issue Board.

Sistema de Priorización MoSCoW

Definición

El método MoSCoW es un marco de priorización utilizado para categorizar los elementos del backlog según su importancia e impacto. MoSCoW significa:

- **Must-Have (Debe Tener):** Requisitos críticos que son innegociables.
- **Should-Have (Debería Tener):** Características importantes que aportan valor significativo, pero no son críticas.
- **Could-Have (Podría Tener):** Características deseables que mejoran el producto, pero no son esenciales.

- **Won't-Have (por ahora) (No Tendrá):** Características fuera del alcance de la iteración actual.

Cómo Funciona

1. **Categorizar Requisitos:** Asignar colaborativamente cada elemento del backlog a una de las categorías de MoSCoW.
2. **Enfocarse en los Must-Haves:** Garantizar que estos se entreguen dentro del sprint o la entrega planificada.
3. **Diferir los Won't-Haves:** Comunicar claramente que estos elementos no forman parte del alcance actual.

Proceso de Aplicación

1. **Refinamiento del Backlog:** Durante las sesiones de refinamiento, revisar y categorizar cada elemento según su valor comercial y urgencia.
2. **Colaboración con los Interesados:** Trabajar estrechamente con los interesados para alinear prioridades y gestionar expectativas.
3. **Discusión en el Equipo:** Compartir prioridades con el Equipo Scrum para garantizar claridad y alineación.

Beneficios

- Garantiza el enfoque en la entrega de las características más críticas primero.
- Evita el desbordamiento del alcance identificando explícitamente lo que no se entregará.
- Proporciona claridad y alineación entre todos los interesados.

Criterios INVEST para Historias de Usuario

Definición

Los criterios INVEST son un conjunto de pautas para crear historias de usuario de alta calidad que sean procesables, claras y valiosas. INVEST significa:

- **Independent (Independiente):** Las historias deben ser autónomas y no depender de otras.
- **Negotiable (Negociable):** Las historias son puntos de partida para la discusión, no especificaciones rígidas.
- **Valuable (Valiosa):** Cada historia debe aportar valor al usuario o al interesado.
- **Estimable (Estimable):** El equipo debe poder estimar el esfuerzo requerido.
- **Small (Pequeña):** Las historias deben ser lo suficientemente pequeñas como para completarse en un sprint.

- **Testable (Verificable):** Las historias deben incluir criterios de aceptación claros.

Elementos Clave

1. **Independiente:** Evitar dependencias entre historias para simplificar la priorización.
2. **Negociable:** Colaborar con los interesados para refinar las historias durante el refinamiento del backlog.
3. **Valiosa:** Escribir historias desde la perspectiva del usuario y resaltar el beneficio.
4. **Estimable:** Proporcionar suficientes detalles para que el equipo estime el trabajo requerido.
5. **Pequeña:** Dividir características más grandes en historias más pequeñas y procesables.
6. **Verificable:** Incluir criterios de aceptación específicos y medibles.

Ejemplos

Buen Ejemplo:

- **Título:** Notificar al usuario sobre baja humedad.
- **Historia de Usuario:**

  Como agricultor, quiero recibir notificaciones cuando la humedad sea baja para poder tomar medidas correctivas.

- **Criterios de Aceptación:**

  Dado que la humedad cae por debajo del umbral, cuando el sistema lo detecta, entonces envía un SMS al usuario.

Mal Ejemplo:

- **Título:** Sistema de notificaciones.
- **Historia de Usuario:**

  Agregar un sistema de notificaciones a la aplicación.

- **Criterios de Aceptación:**

  Ninguno proporcionado.

Integración al Flujo de Trabajo

Cómo Trabajaremos Juntos

1. **Priorización del Backlog:** Usar el método MoSCoW durante el refinamiento para alinear prioridades.
2. **Redacción de Historias de Usuario:** Escribir historias utilizando los criterios INVEST para garantizar claridad y alineación.

3. **Refinamiento Colaborativo:** Involucrar regularmente al Equipo Scrum y a los interesados para refinar los elementos del backlog.
4. **Comunicación Transparente:** Compartir prioridades y avances de manera abierta con todos los miembros del equipo y los interesados.

Herramientas y Prácticas

- **Gestión del Backlog:** Usar Gitex para rastrear y organizar tareas.
- **Sesiones Colaborativas:** Dos reuniones stand-up por semana (miércoles y viernes). Un día en oficina (miércoles).
- **Documentación Clara:** Mantener documentación actualizada y accesible para los elementos del backlog y las prioridades.

3 Arquitectura de Software y Redes. Diseño de HMI del Sistema

En este capítulo se pretende explicar la arquitectura del frontend, la arquitectura de red y el diseño del HMI, para dar un mejor contexto y visión general de todo el sistema.

No se analizará la implementación técnica ya que yo no he sido la persona que lo ha realizado, habiendo participado únicamente en las decisiones de diseño y en la especificación de los requisitos.

Para el proyecto del sistema mecatrónico inteligente para la germinación de semillas ha sido sumamente importante realizar un software de calidad, estructurado, organizado y modularizado. Por lo tanto, siempre le hemos dado una gran importancia al diseño de la arquitectura del software y de red, tratando siempre de diseñar sus bases de una forma robusta y escalable, para que nos permitieran ir añadiendo mejoras y diferentes módulos.

También ha sido un aspecto clave diseño de la arquitectura de red, ya que en este proyecto existen distintos dispositivos que ejecutan los diferentes módulos del sistema, por lo tanto, es primordial que las comunicaciones estén bien diseñadas y estructuradas.

Por otro lado, desde el principio fue nuestro objetivo desarrollar una interfaz para que el usuario pudiera visualizar y controlar el sistema al completo de forma centralizada, intuitiva y simple. Este HMI, además de permitir al usuario visualizar y obtener la información del estado de las distintas partes del sistema, le permiten controlarlas de una forma manual clásica, a través de un ordenador o una pantalla táctil, pero también le permiten controlar el sistema por métodos avanzados de control, como, por ejemplo, lenguaje natural, y en un futuro, control por gestos.

3.1 Arquitectura de Software

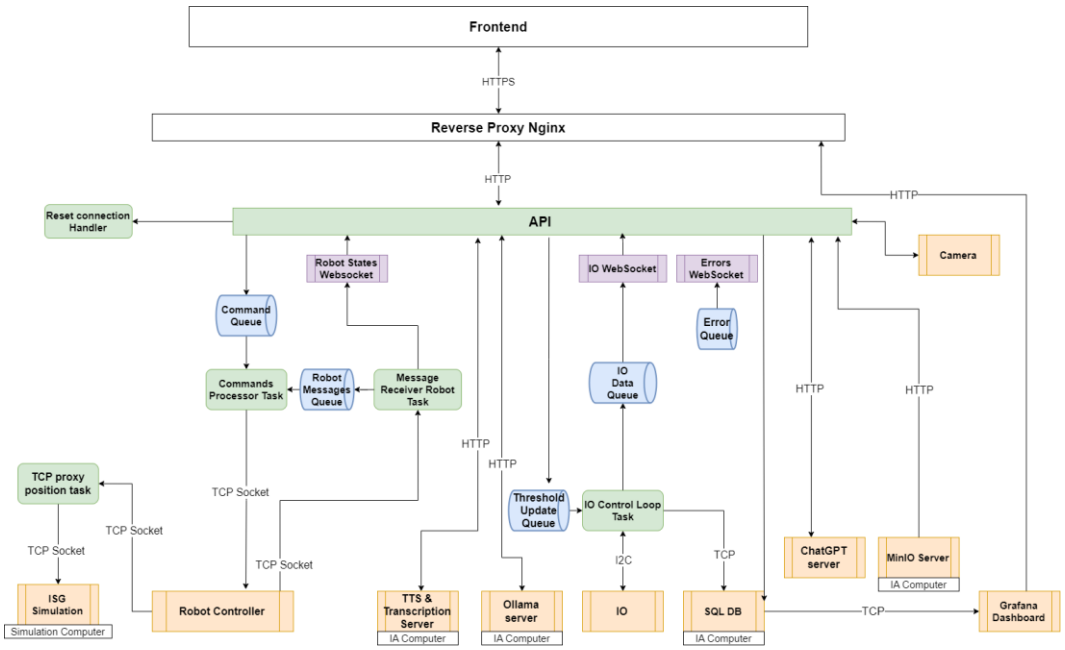

Ilustración 11: Diagrama de la arquitectura del software del sistema mecatrónico para la germinación de semillas.

3.1.1 Descripción del software

El software sigue una arquitectura basada en eventos, donde una API REST asíncrona sirve como controlador central para todas las operaciones e interacciones con diversos componentes del sistema internos y externos (Ilustración 11, que se encuentra ampliado en el Anejo 4). Tareas asincrónicas diferentes gestionan subsistemas críticos, como la comunicación del robot y el control de I/O. Estas tareas se comunican entre sí y con el servidor de API REST utilizando colas y eventos.

Una tarea responsable de traducir los datos de posición para la simulación del ISG se ejecuta en un proceso separado debido a su carga computacional alta. Ejecutarlo en un proceso diferente nos permite aprovechar la paralelización mediante el uso de múltiples núcleos de CPU. A pesar de estar en un contexto de ejecución distinto, esta tarea permanece asíncrona, pero opera dentro de un bucle de eventos asincrónicos distinto.

3.1.2 Frontend y Reverse Proxy

El frontend se comunica con el backend a través de solicitudes HTTPS, con un proxy inverso (Nginx) manejando la terminación SSL, el filtrado y la distribución de las solicitudes entrantes a los servicios apropiados (Grafana, MinIO, WebSockets, API REST). El proxy también asegura que se use la versión correcta de HTTP para cada servicio.

Se utilizan WebSockets para la comunicación en tiempo real desde el backend al frontend, permitiendo notificaciones impulsadas por eventos sin requerir solicitudes de cliente explícitas. Este mecanismo se usa para enviar actualizaciones sobre errores, nuevos datos de E/S y estados del robot.

3.1.3 Integración de Componentes en el Backend

El backend integra diversos componentes internos y externos, incluyendo bases de datos y servidores externos (por ejemplo, transcripción, TTS, MinIO), utilizando diferentes métodos de comunicación:

- **Solicitudes HTTP** se usan para comunicarse a través de una interfaz con servicios externos a través de la API REST.
- **Sockets TCP asíncronos (ASockets)** se emplean para la comunicación con el controlador del robot y para transmitir datos de posición a la simulación del robot.

Este enfoque modular asegura un flujo de datos eficiente y una respuesta rápida en todo el sistema.

3.2 Arquitectura de Red

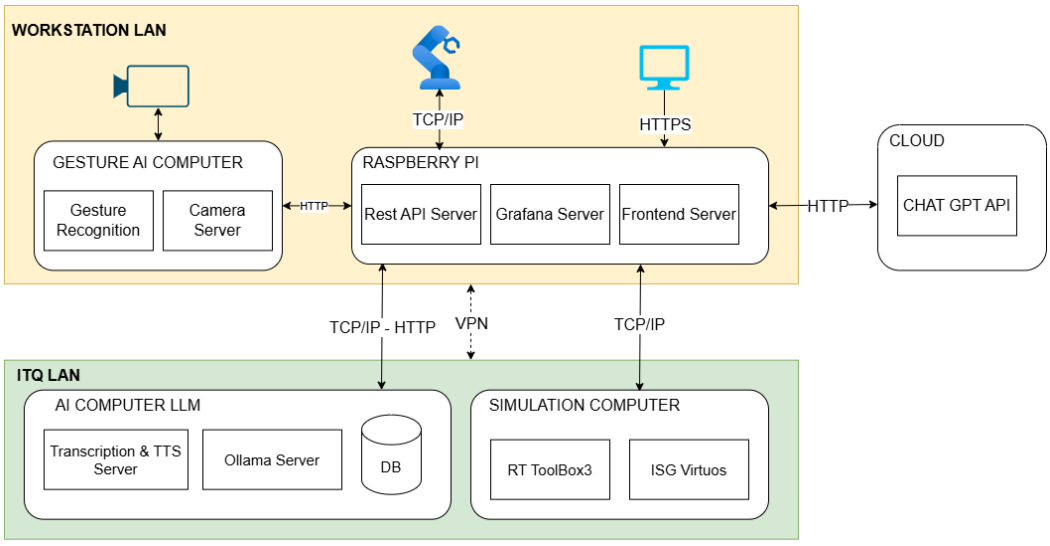

Ilustración 12: Arquitectura de red del sistema mecatrónico para la germinación de semillas

3.2.1 Resumen de Conexión del Sistema

El diagrama de la ilustración 12 representa cómo se interconectan los componentes del sistema. El sistema opera como una arquitectura de software distribuido, que abarca dos redes locales distintas dentro de la empresa: la **LAN de Estación de Trabajo** y la **ITQ LAN**.

3.2.2 LAN de Estación de Trabajo

Esta red conecta los dispositivos ubicados dentro del demostrador, incluyendo:

- **Computadora de IA de Gestos**
- **Raspberry Pi**
- **Robot**
- **Panel de Control** (Dispositivo usado por el operador para interactuar con el sistema a través de una interfaz web)

La Computadora de IA de Gestos alberga:

- Un **Servidor de Cámara**, que transmite video desde la cámara de profundidad.
- Un **Servidor de Reconocimiento de Gestos de IA**, responsable del procesamiento de entradas de gestos.

Estos servidores se pueden acceder a través de HTTP desde el Panel de Control (a través de un navegador web) y la Raspberry Pi cuando sea necesario.

El Raspberry Pi ejecuta múltiples servicios críticos:

- **Servidor de API REST**: Facilita la comunicación con el robot a través de sockets TCP/IP.
- **Servidor de Grafana**: Para visualizar datos del sistema.
- **Servidor Frontend**: Proporciona la interfaz web para el Panel de Control.

Además, el **Servidor de API REST** en la Raspberry Pi:

- Se conecta al servicio de nube ChatGPT a través de HTTP para externalizar tareas de reconocimiento de imágenes.
- Interactúa con diversos servicios alojados en computadoras en la ITQ LAN, lo que requiere conectar ambas redes mediante un **VPN**.

  Nota: Una decisión está pendiente sobre si enrutar todo el tráfico VPN a través de un solo dispositivo o conectar cada dispositivo de la LAN de Estación de Trabajo individualmente al VPN.

3.2.3 ITQ LAN

La ITQ LAN es la red interna de la empresa, donde dos computadoras de alto rendimiento manejan las tareas más intensivas del sistema: Simulación y Procesamiento de IA.

- **Computadora de Simulación**:
  - Ejecuta RT Toolbox e ISG Virtuos para simulación del robot.
  - Se comunica con el Raspberry Pi a través de sockets TCP/IP sobre el VPN.
- **Computadora de IA**:

- Aloja el Servidor de Transcripción y Síntesis de Voz (TTS), accesible desde el backend en el Raspberry Pi a través de HTTP.
- Ejecuta un Servidor Ollama para servir LLMs (Modelos de Lenguaje Grandes) como servicio, también accesible vía HTTP.
- Almacena datos del sistema mediante:
  - Una Base de Datos PostgreSQL para datos estructurados.
  - Un Servidor MinIO para almacenar y recuperar imágenes.

Esta arquitectura distribuida asegura un procesamiento y comunicación de datos eficientes mientras aprovecha los recursos computacionales de alto rendimiento.

3.3 HMI

En este capítulo se expone el diseño del HMI y se explica cómo debe ser utilizado.

El HMI ha sido programado como el frontend de una aplicación web, tal y como se ha mencionado en los apartados anteriores.

Analizaremos diferentes ventanas y funcionalidades paso a paso, sin analizar cómo ha sido implementado a un nivel técnico, ya que you no he sido la persona encargada de implementar el software, pero sí que he participado en las tareas de diseño.

3.3.1 Barra Superior

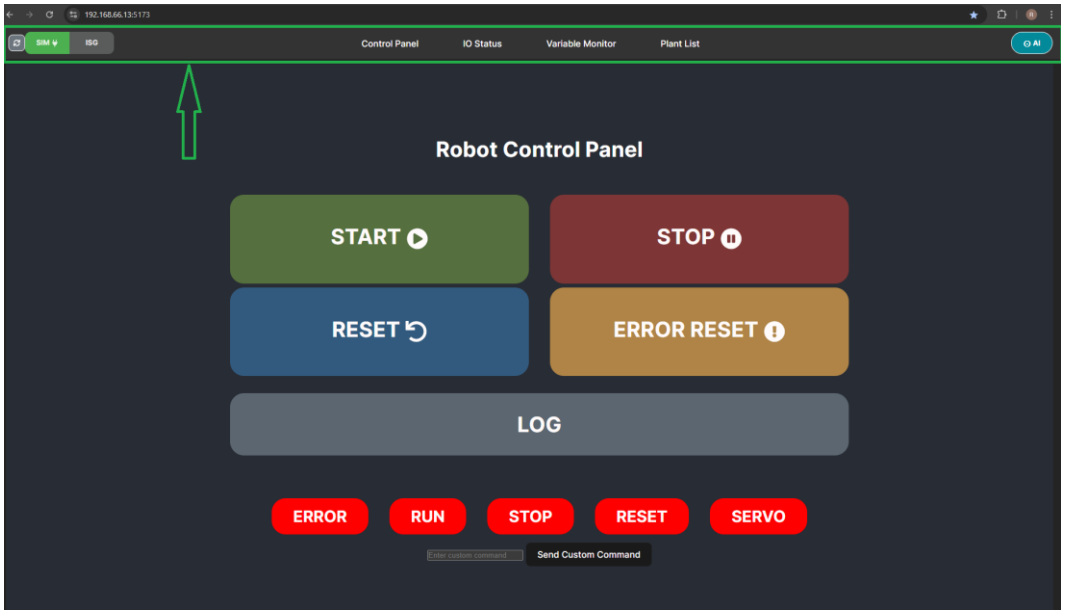

Ilustración 13: Barra superior del frontend

En la parte superior del frontend, puedes encontrar la Barra Superior (Ilustración 13). En la sección central de la barra (Ilustración 14), puedes

seleccionar la pestaña o ventana a la que deseas dirigirte entre las siguientes: Panel de Control, Estado de Entradas/Salidas, Monitor de Variables, Lista de Plantas.

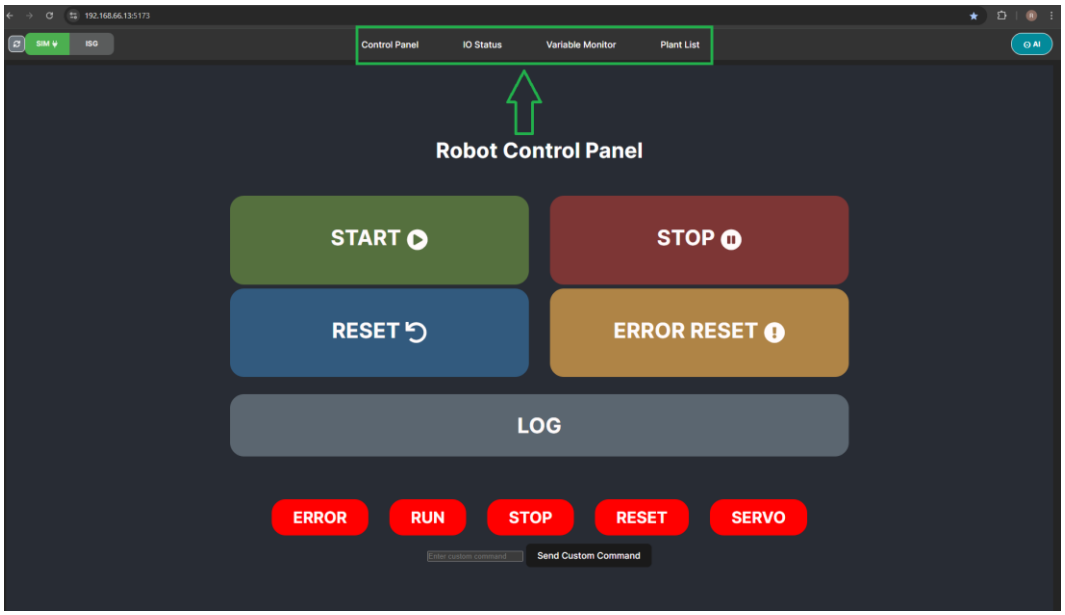

Ilustración 14: Selección de ventanas

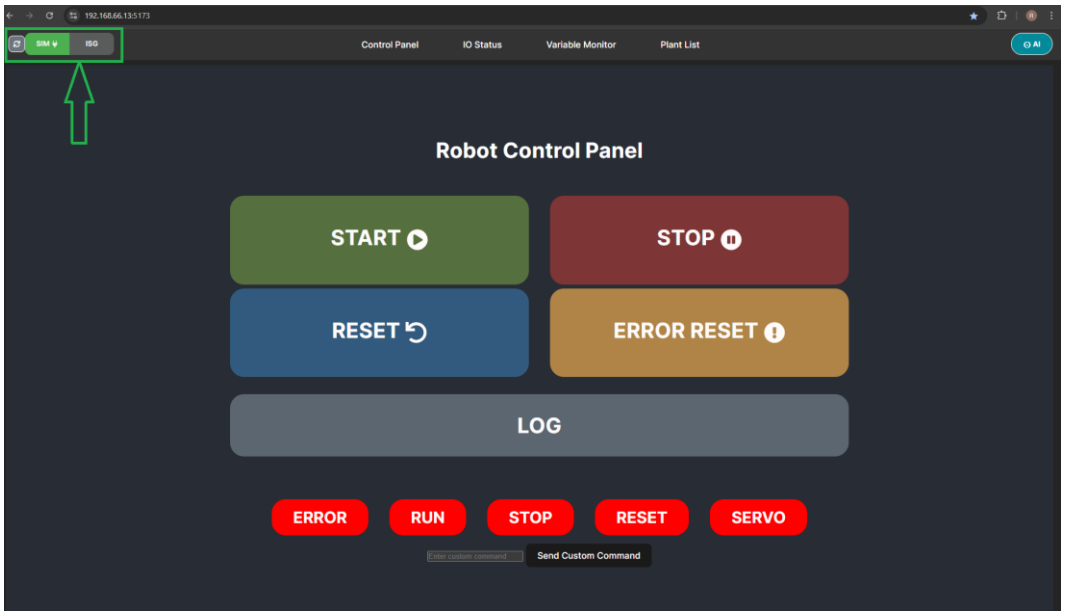

Ilustración 15: Display del estado de conexión

A la izquierda, encontrarás el Display de Estado de Conexión (Ilustración 15), que informa sobre el estado actual de la conexión y un botón con dos flechas formando un círculo para reiniciar la conexión.

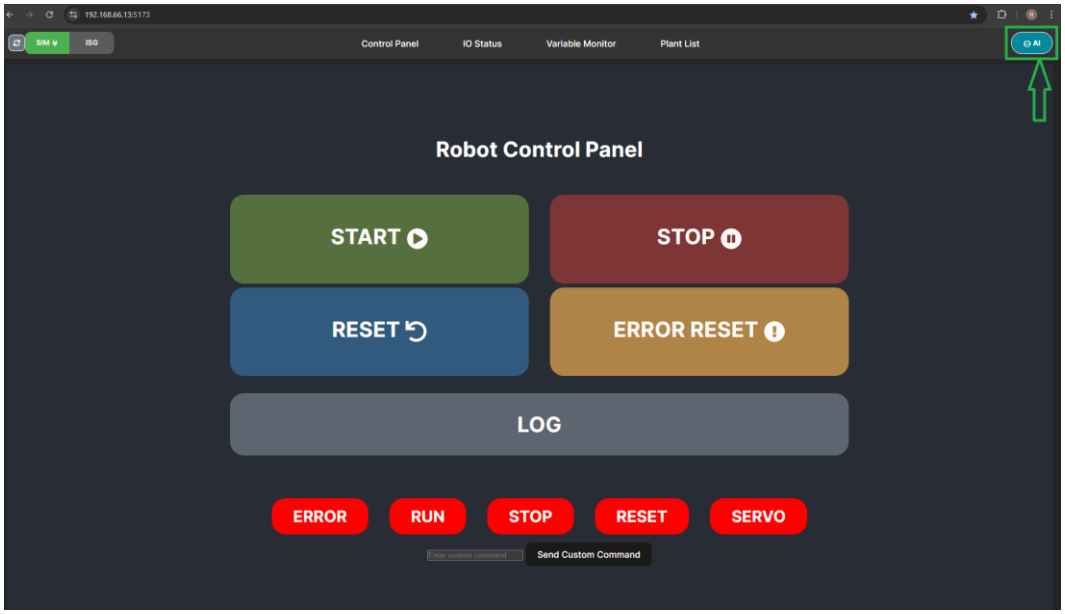

Ilustración 16: Botón para activar el asistente para el control por voz

En la parte derecha de la barra de pestañas, hay un botón azul con la palabra AI (Ilustración 16). Este botón se utiliza para iniciar el Asistente de Voz IA, que permite al usuario controlar el sistema con comandos de voz en lenguaje natural.

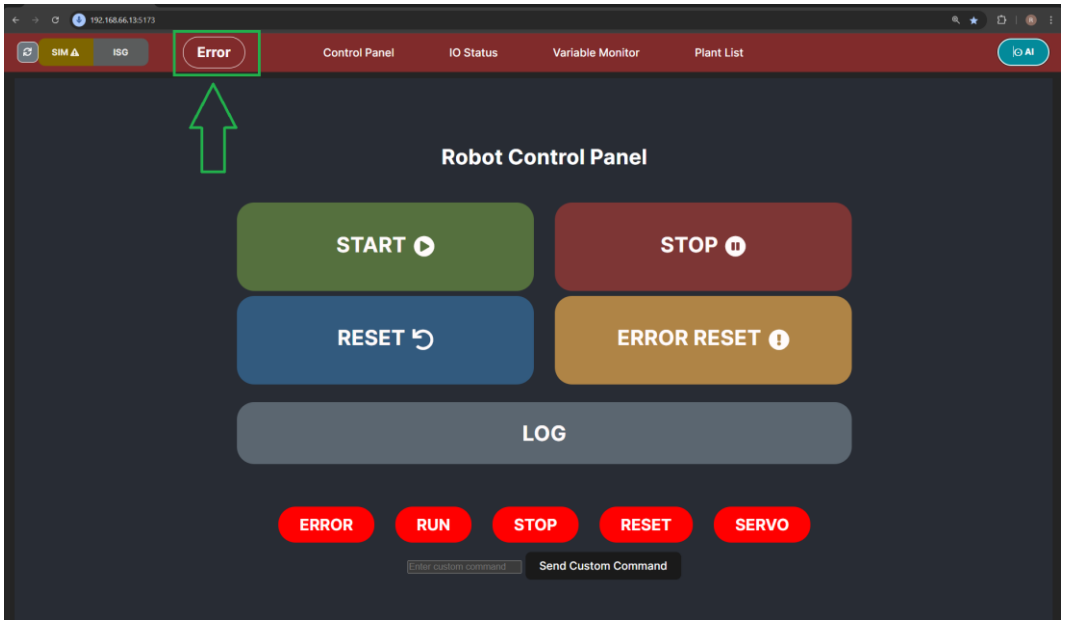

Ilustración 17: Botón para gestionar errores

Además, si ocurre un error, la Barra Superior parpadeará de color rojo y aparecerá un nuevo botón entre el Display de Estado de Conexión y la selección de ventana. Este botón se utiliza para desplegar la Ventana de Manejo de Errores (Ilustración 17).

3.3.2 Display de Estado de Conexión

En la parte izquierda de este display, hay un botón con dos flechas formando una forma circular, que permite reiniciar la conexión. Es importante notar que el modo de conexión es seleccionado automáticamente por la Raspberry Pi siguiendo una jerarquía de prioridades: la conexión con el robot y la funcionalidad espejo del Gemelo Digital es la más alta, la conexión con la simulación del Gemelo Digital es intermedia, y sin conexión es la más baja. Esto significa que si el estado actual de conexión es 'Simulación' y conectas el robot y presionas el Botón de Reinicio de Conexión, el modo de conexión cambiará a Robot.

El display está dividido en dos partes: en el lado izquierdo del display, donde en la imagen puedes ver 'SIM' en verde, informa sobre el estado del modo de conexión. En el lado derecho del display, donde dice 'ISG', informa si la conexión con el software de simulación (ISG Virtuos) está establecida; el color gris, como se ve en la foto, significa que ISG no está disponible, y el color azul significaría que el software del Gemelo Digital está conectado.

3.3.3 Botón Asistente de Voz IA

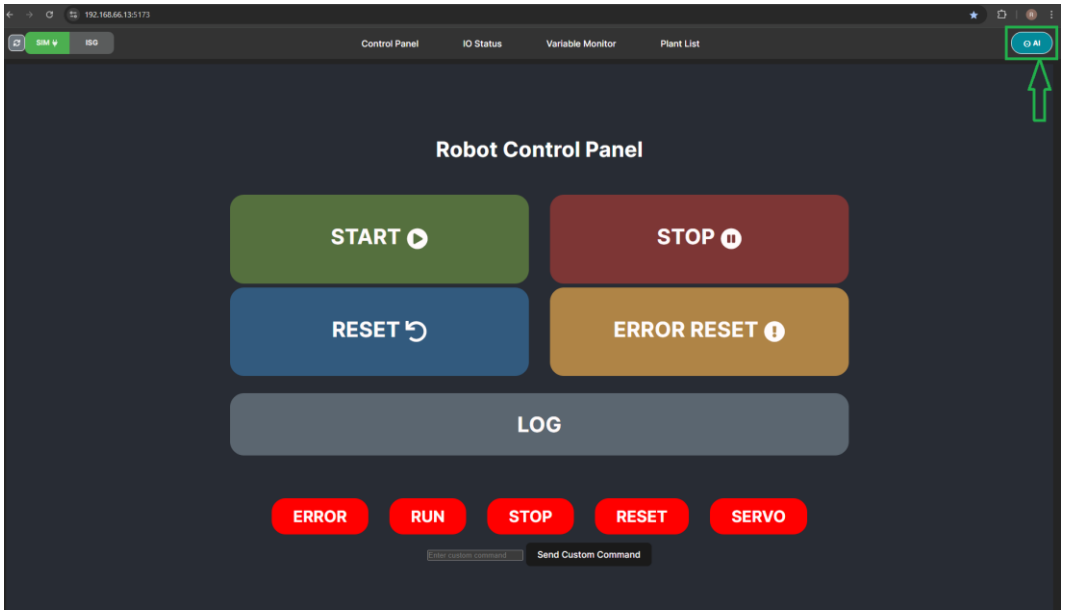

Ilustración 18: Botón AI

Cuando presionas el botón AI (Ilustración 18), tendrá lugar una animación en la cual este botón se hará más ancho y mostrará la palabra 'Grabando' con un botón cuadrado y un botón de basura a la izquierda (Ilustración 19). Cuando se muestra el texto 'Grabando', el sistema está grabando audio para recibir los comandos de voz en lenguaje natural del usuario. Después de hablar, tienes dos opciones: hacer clic en el botón de basura, y los comandos de voz grabados

serán eliminados, o presionar el botón cuadrado, por lo que los comandos de voz se enviarán para su procesamiento.

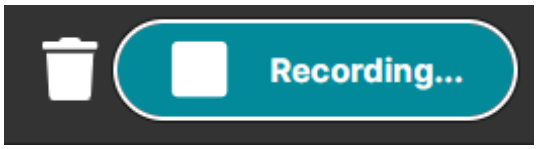

Ilustración 19: Animación del botón AI cuando se está grabando el audio

Una vez que se envía el comando de voz después de presionar el botón cuadrado, es procesado por el software de transcripción (Whisper), y una vez transcrita, se muestra en el Display de IA (Ilustración 20). Mientras se muestra la transcripción, también se envía al LLM para su procesamiento.

Ilustración 20: Transcripción del audio grabado reproducido en el display del botón AI

Una vez que el LLM procesa la solicitud en lenguaje natural, responderá con un comando, que se mostrará en el Display de IA durante dos segundos (Ilustración 21), permitiendo al usuario cancelar el comando haciendo clic en la basura.

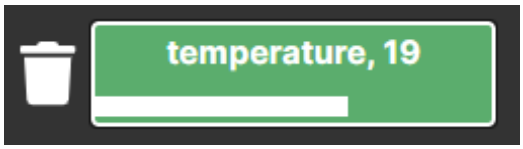

Ilustración 21: Comando generado por el LLM mostrándose en el display del botón AI

Si la ejecución del comando no es instantánea, como mover el robot, el display se volverá amarillo y mostrará 'Ejecutando' mientras el robot se mueve.

3.3.4 Botón de Manejo de Errores

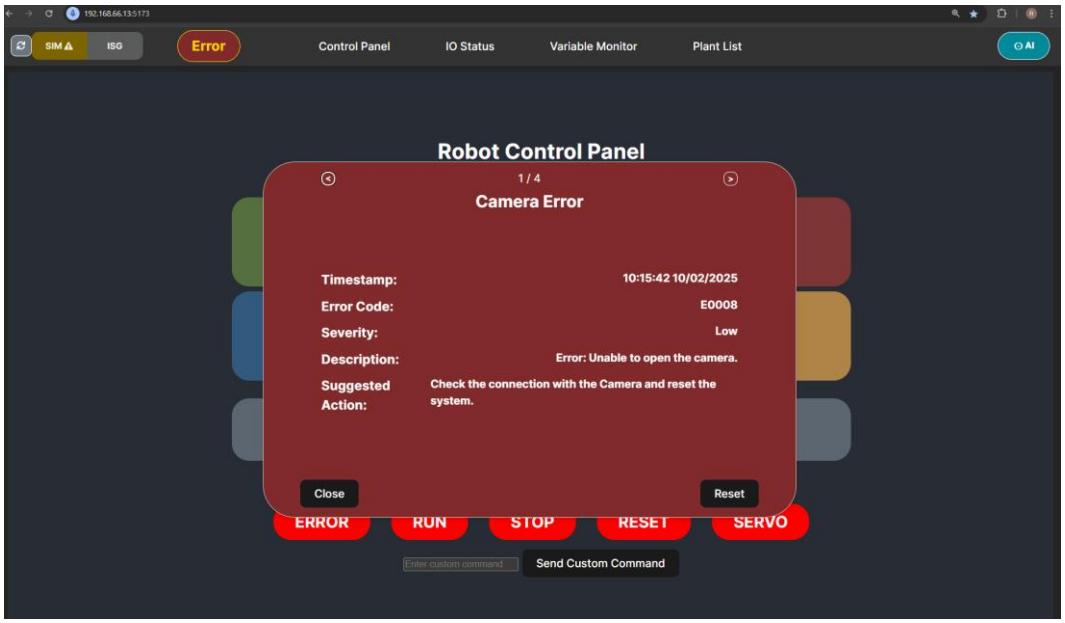

Ilustración 22: Panel de manejo de errores

Una vez que ocurre un error, aparecerá el Botón de Manejo de Errores (Ilustración 17) entre el Display de Estado de Conexión y las pestañas de cambio de ventana. Una vez que hagas clic en este botón, se desplegará la Ventana de Manejo de Errores (Ilustración 22), donde puedes ver qué tipo de error está ocurriendo y cómo solucionarlo. Esta ventana permite al usuario navegar entre errores, si hay más de uno. Además, esta ventana te permite restablecer los errores haciendo clic en el botón 'Restablecer' o simplemente cerrar la ventana haciendo clic en 'Cerrar'.

3.3.5 Panel de Control del Robot

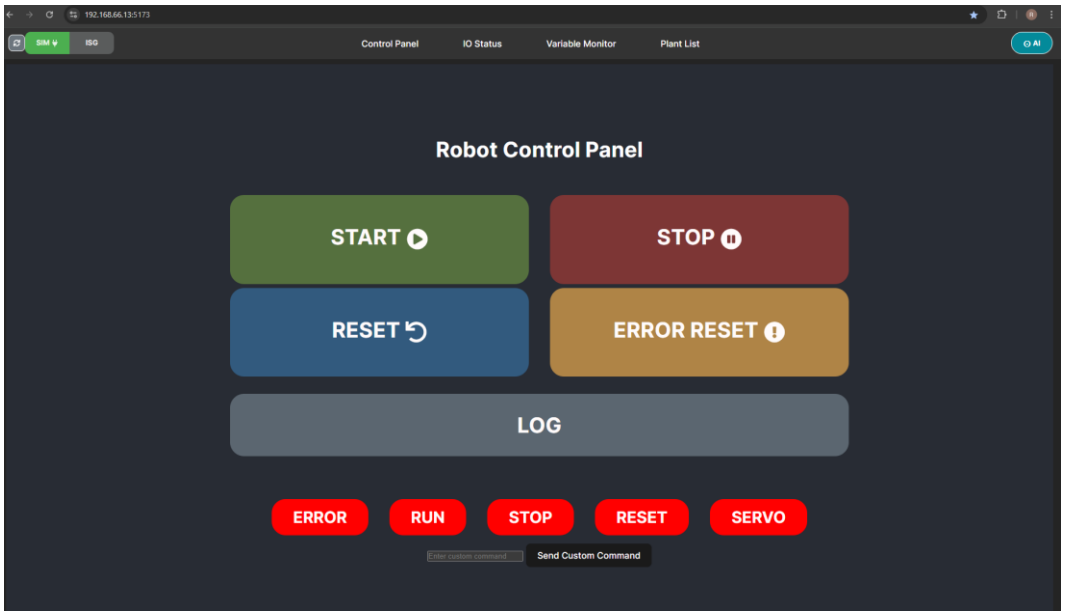

Ilustración 23: Panel de control del robot

Este panel (Ilustración 23)muestra el estado del robot y permite al usuario controlarlo directamente.

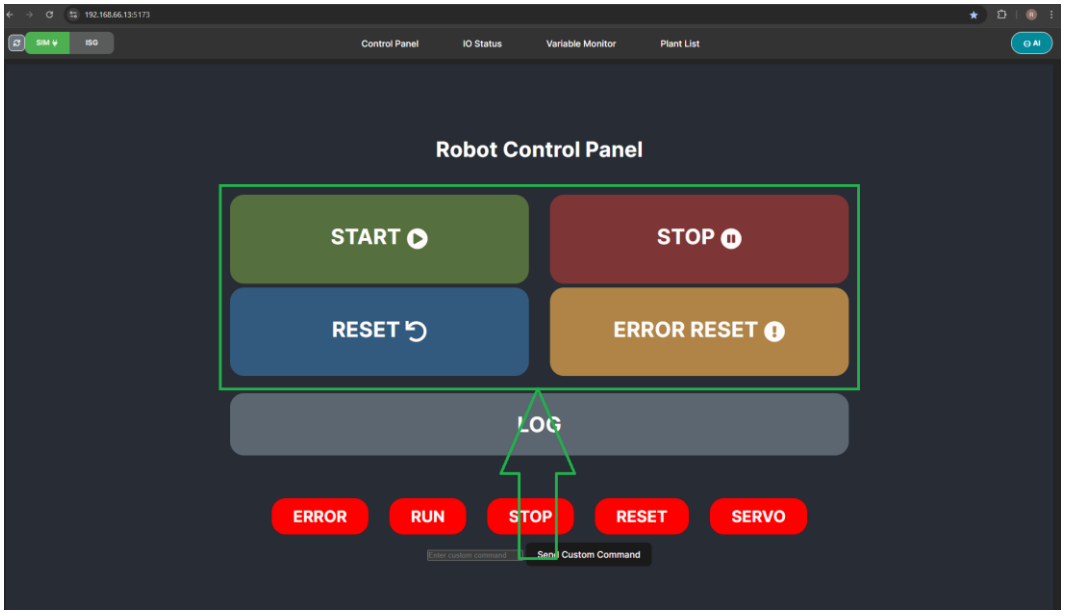

Ilustración 24: Botones de control del robot

En la parte superior, hay cuatro botones (Ilustración 24) que desencadenan diferentes acciones en el controlador del robot. El botón verde se utiliza para iniciar el programa almacenado en la primera ranura del controlador. El botón rojo se utiliza para detener el programa en ejecución en la primera ranura del controlador del robot. El botón azul se utiliza para reiniciar el programa del robot en la primera ranura del controlador. Finalmente, el botón amarillo se

utiliza para restablecer cualquier error que pueda ocurrir en el controlador del robot.

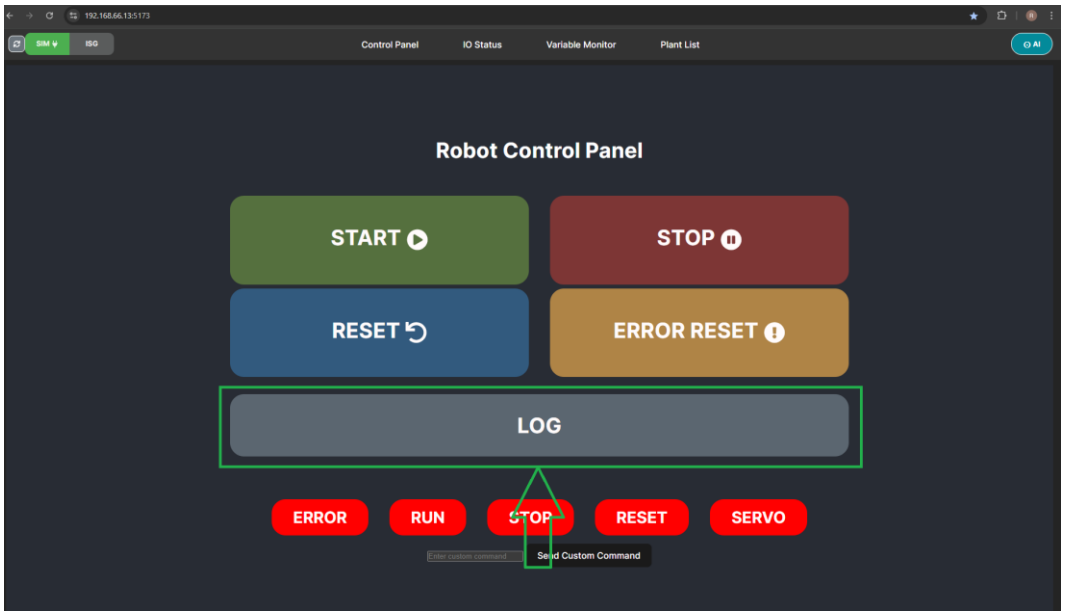

Ilustración 25: Botón para desplegar el panel del registro de la comunicación con el robot

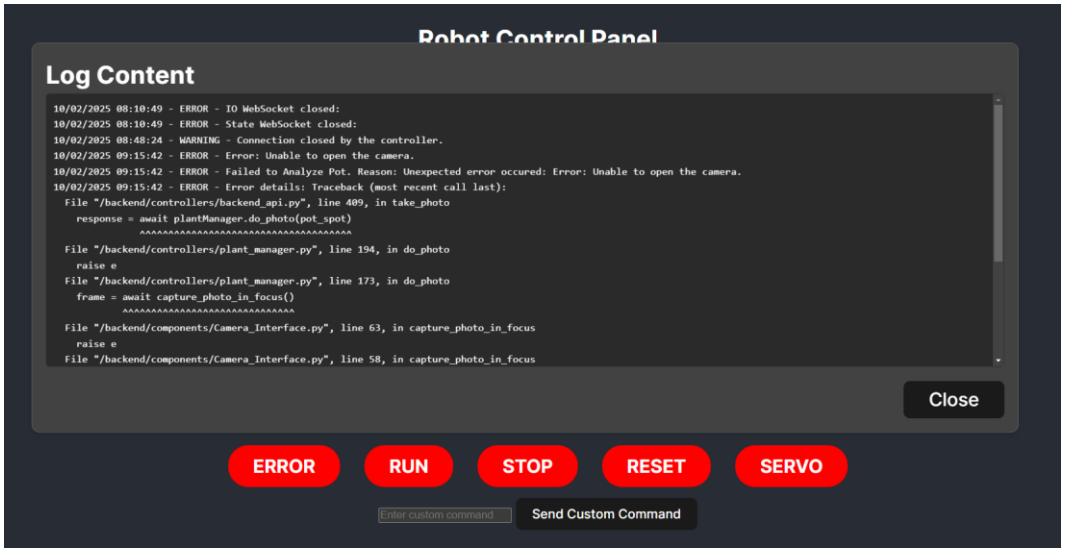

Ilustración 26: Panel que muestra el contenido del registro de las comunicaciones con el robot

Inmediatamente debajo de estos cuatro botones, hay un botón gris (Ilustración 25) que se usa para desplegar el registro del robot (Ilustración 26), donde el usuario puede navegar por la información generada por el servidor sobre la comunicación con el robot.

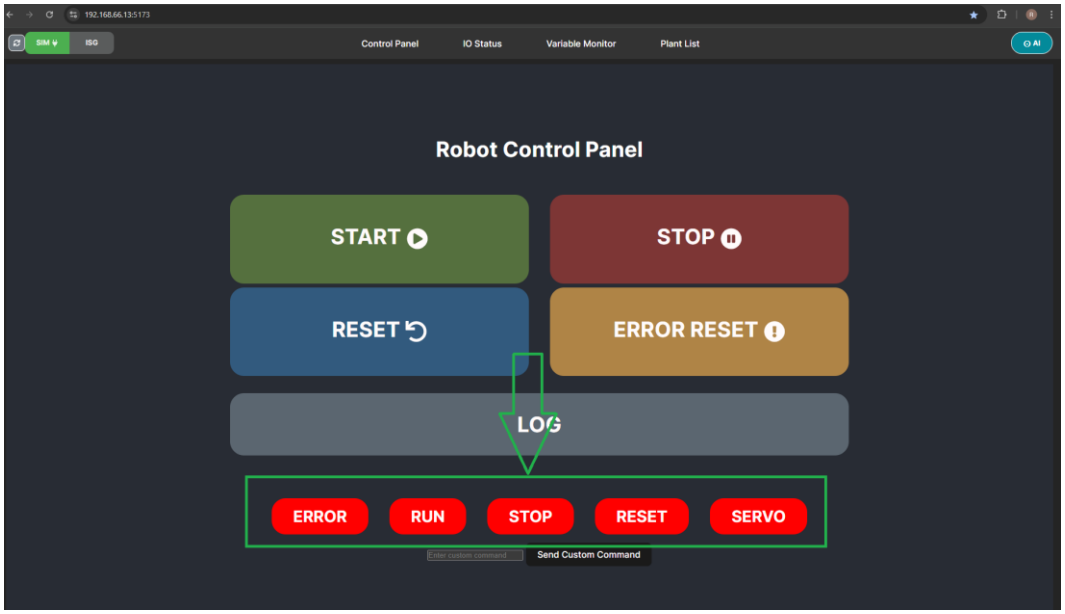

Ilustración 27: Displays informativos sobre el estado del robot

Debajo del botón de registro, hay cinco displays que informan al usuario sobre el estado del robot (Ilustración 27). Los displays muestran variables internas booleanas del controlador del robot, con rojo indicando falso y verde indicando verdadero. El display de error indica si hay algún error en el controlador, el display de ejecución indica si el programa en la primera ranura está en ejecución, el display de detención indica si el programa en el primer slot está detenido, el display de reinicio indica si el programa en el primer slot se ha reiniciado y el display de servomotor indica si los servomotores están encendidos.

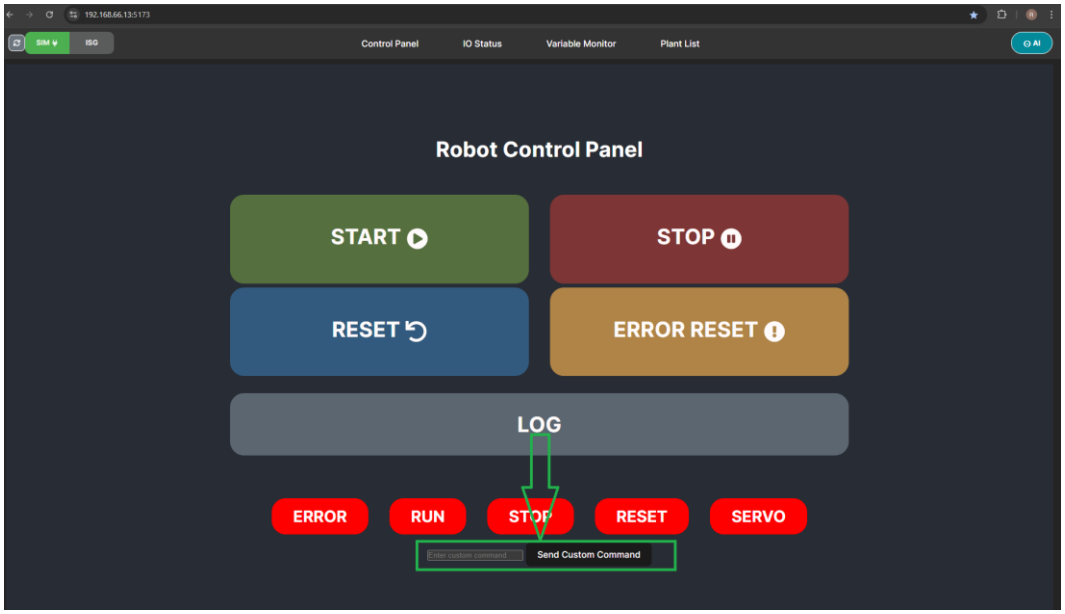

Ilustración 28: Campo para enviar comandos personalizados

Justo debajo de las pantallas, hay un campo de texto (Ilustración 28) donde el usuario puede enviar comandos personalizados al controlador, en lugar de usar los comandos fijos que permiten los botones. El operador solo necesita escribir el comando deseado y enviarlo haciendo clic en el botón 'Enviar Comando Personalizado'.

3.3.6 Estado de Entradas/Salidas

En esta ventana de entradas y salidas (Ilustración 29), el usuario puede observar tanto los valores de los sensores y actuadores como controlar a los actuadores.

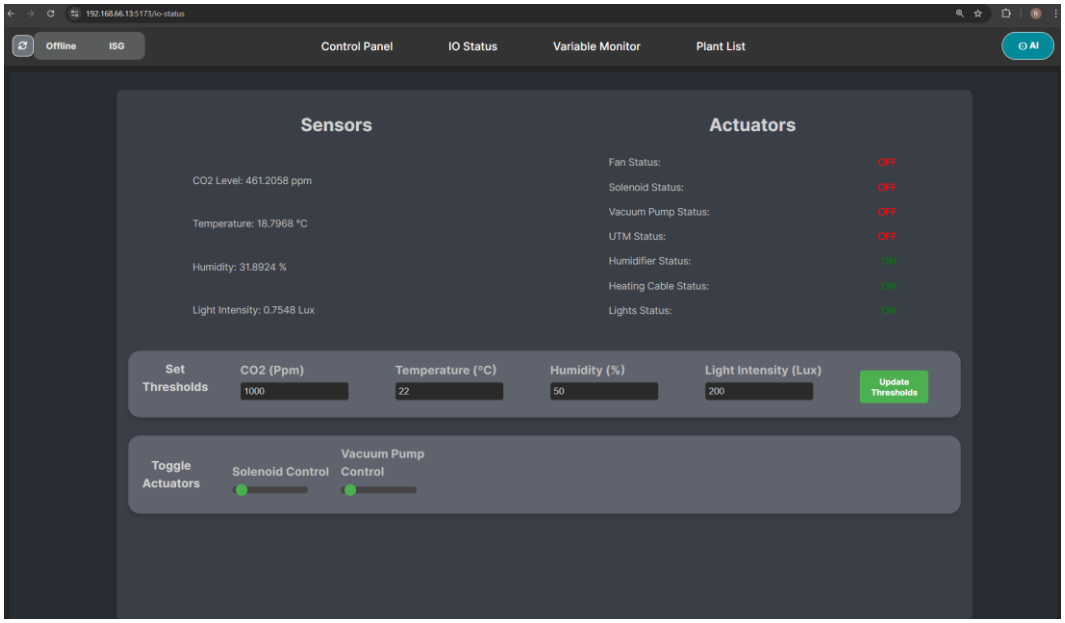

Ilustración 29: Ventana de entradas y salidas

En la parte superior, hay una tabla donde el usuario puede ver el estado de los diferentes sensores y actuadores. En el lado izquierdo de la lista (Ilustración 30), puedes ver los valores exactos del nivel de CO2, temperatura, humedad e intensidad lumínica proporcionados por los sensores.

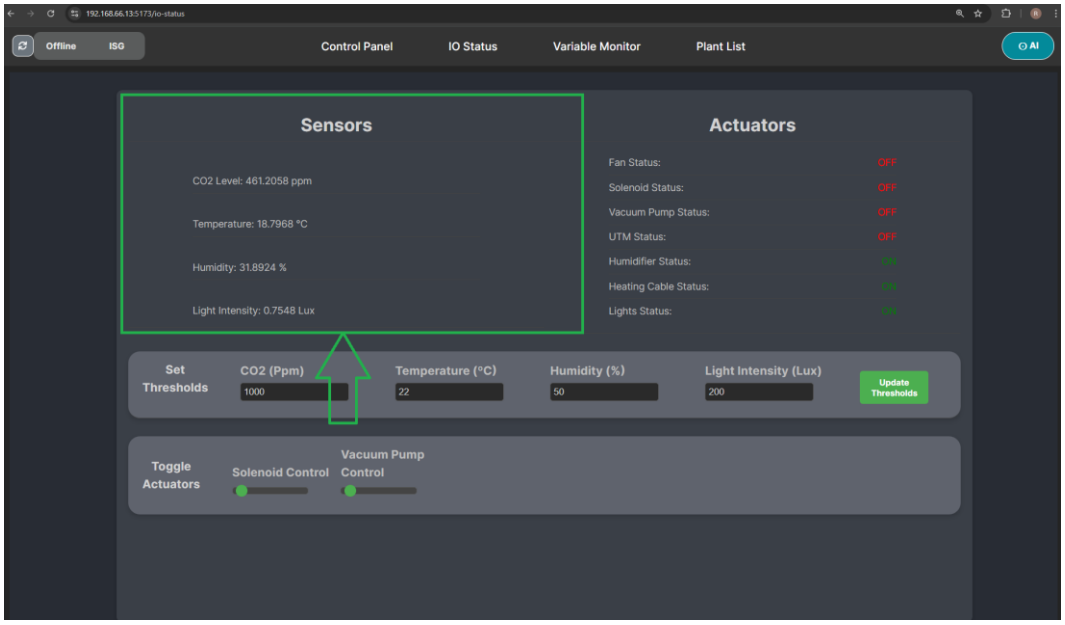

Ilustración 30: Lista de valores de los sensores en tiempo real

En el lado derecho de la tabla, puedes observar el estado de los actuadores, que siempre son booleanos (Ilustración 31).

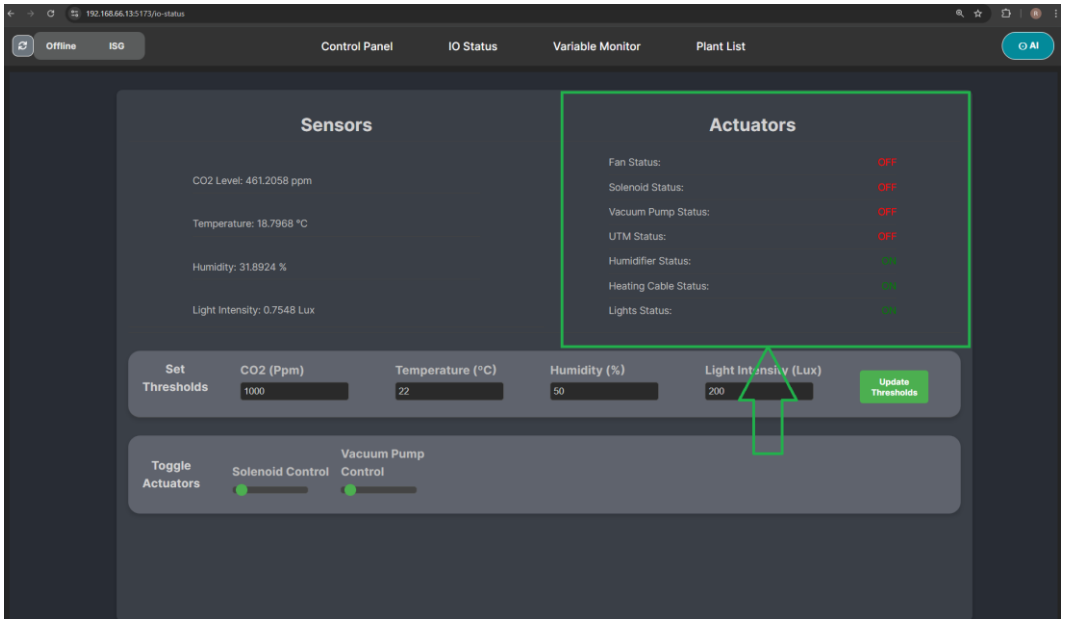

Ilustración 31: Lista del estado de los actuadores en tiempo real

Justo debajo de la lista, encontrarás el menú 'Establecer Umbrales' (Ilustración 32), donde el usuario puede establecer los valores deseados para las variables medidas por los sensores. Una vez que el usuario cambie los valores objetivo, se debe presionar el botón 'Actualizar Umbrales', de modo que el bucle de control automático actualice los valores objetivo de las variables y comience a controlar los actuadores según ellos.

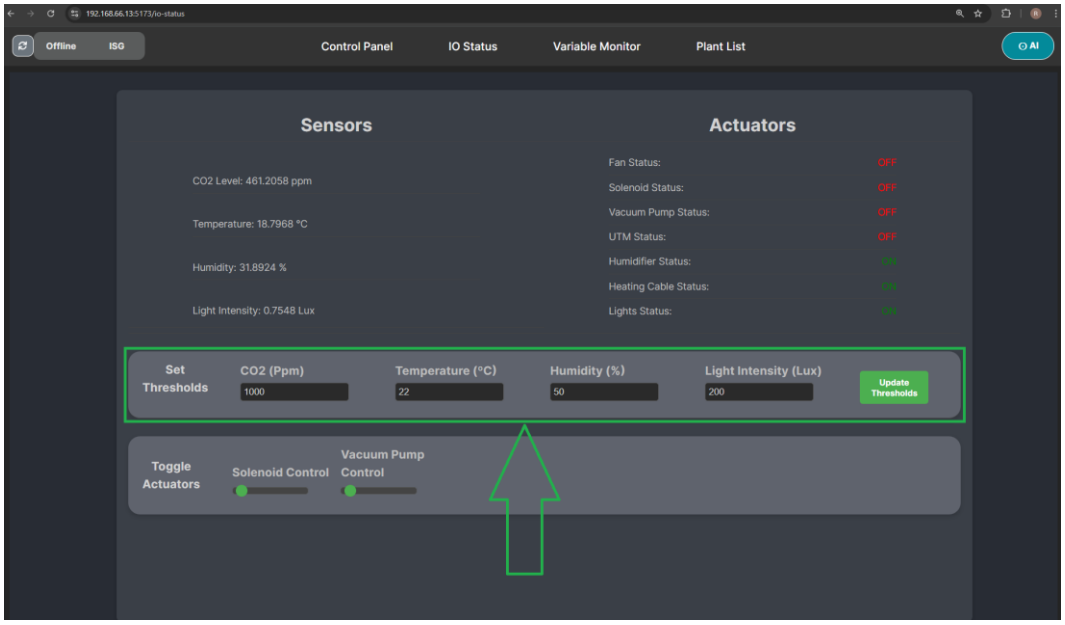

Ilustración 32: Menú para establecer los umbrales para los valores de las variables controladas

Finalmente, en la parte inferior de la ventana, está el menú 'Alternar Actuadores' (Ilustración 33), donde el usuario puede controlar manualmente los actuadores que no están relacionados con las variables ambientales del módulo aislado, como la válvula electromagnética, utilizada para riego, o la bomba de vacío, utilizada para recoger semillas.

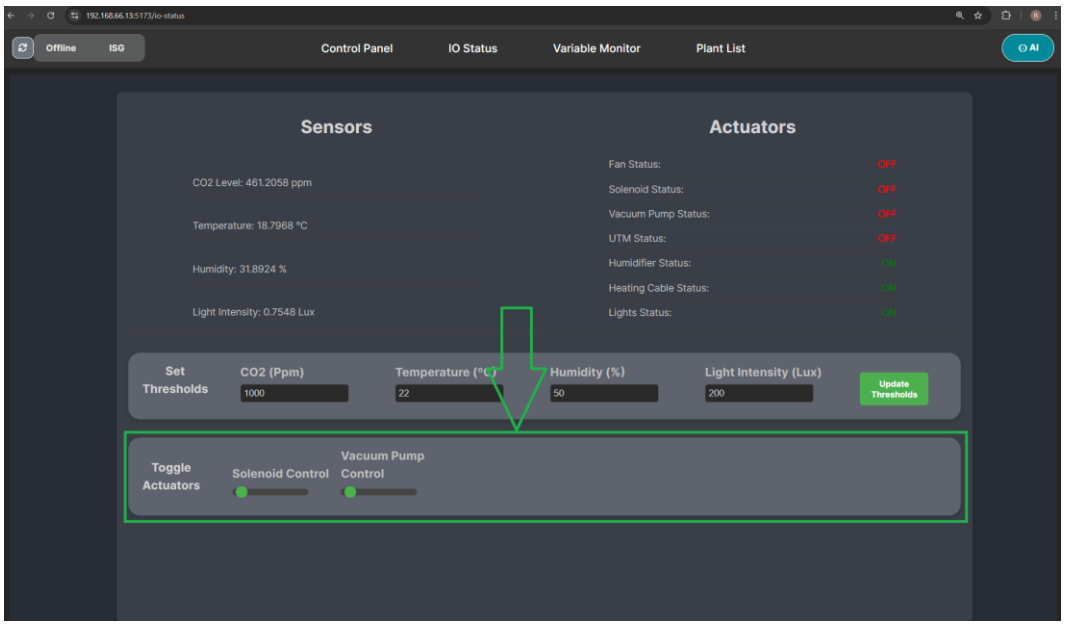

Ilustración 33: Campo para alternar el estado de los actuadores no relacionados con las variables controladas

3.3.7 Monitor de Variables

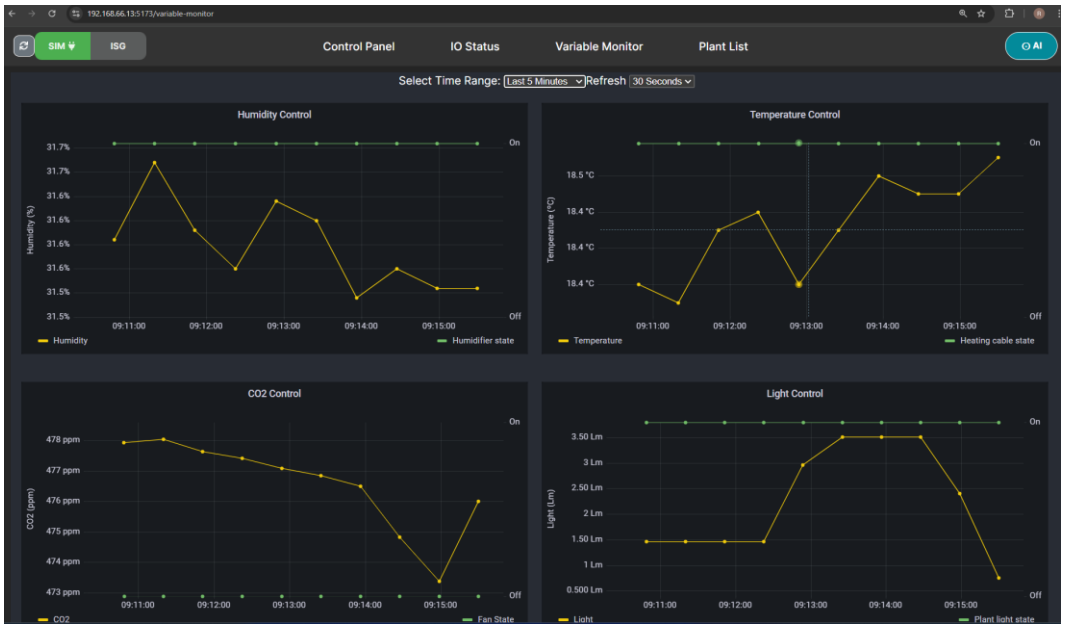

Ilustración 34: Panel con el monitor de variables

En este panel (Ilustración 34), el usuario puede observar visualmente los gráficos de los datos históricos de los sensores y sus correspondientes actuadores. Los gráficos se han hecho con Grafana usando los datos de los sensores y actuadores almacenados en la base de datos.

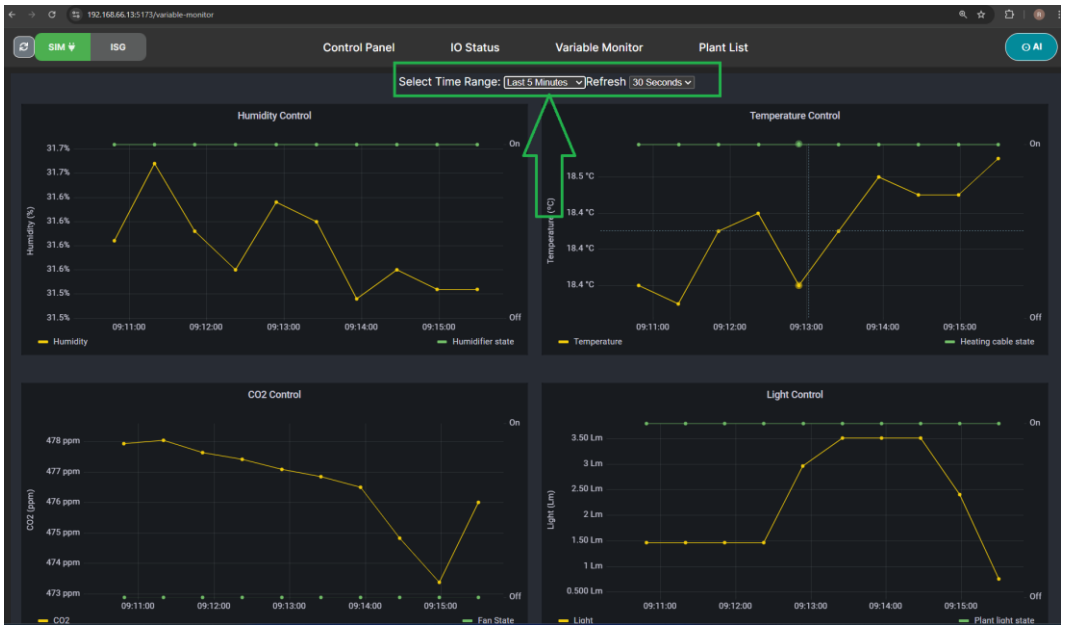

Ilustración 35: Menú desplegable del panel de monitoreo

En la parte superior, hay un menú desplegable (Ilustración 35) donde el usuario puede seleccionar el período de tiempo durante el cual se deben mostrar los datos, que va desde los últimos 5 minutos hasta los siete días pasados. Aquí

también puedes seleccionar la tasa de actualización de los gráficos en otro menú desplegable, que va desde sin actualización a una actualización cada 30 segundos.

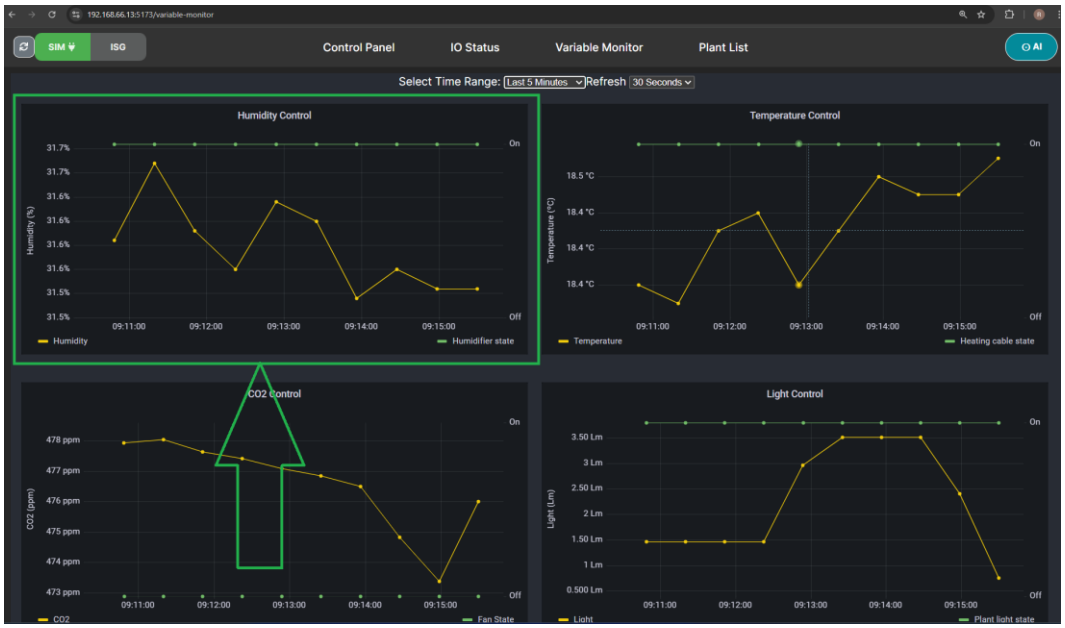

Ilustración 36: Gráfica con el valor de la humedad y el estado del humidificador

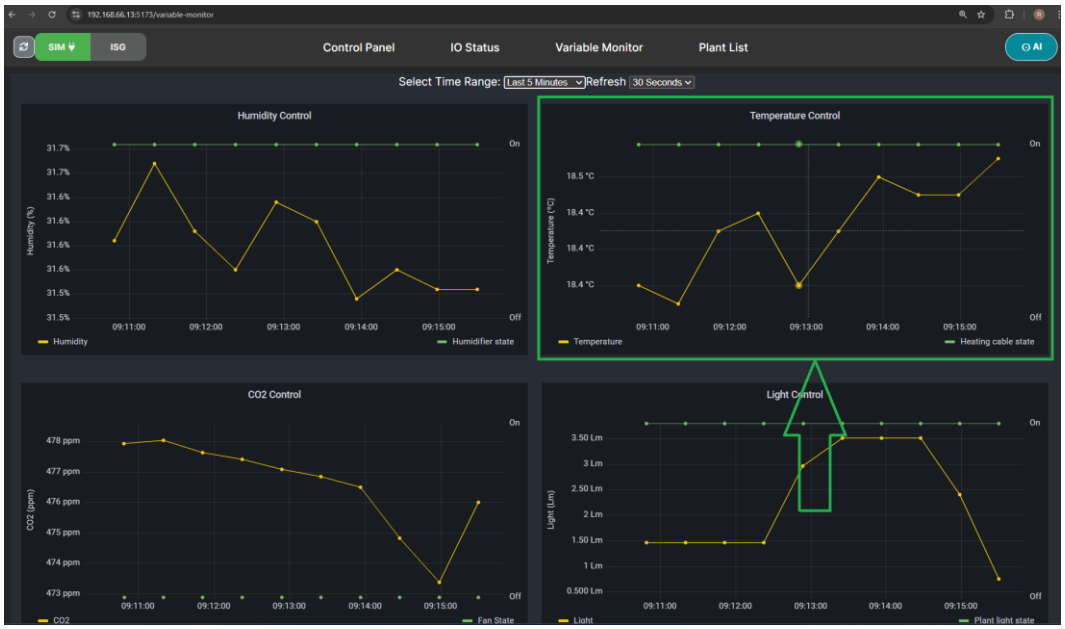

Ilustración 37: Gráfica con el valor de la temperatura y el estado del cable calorífico

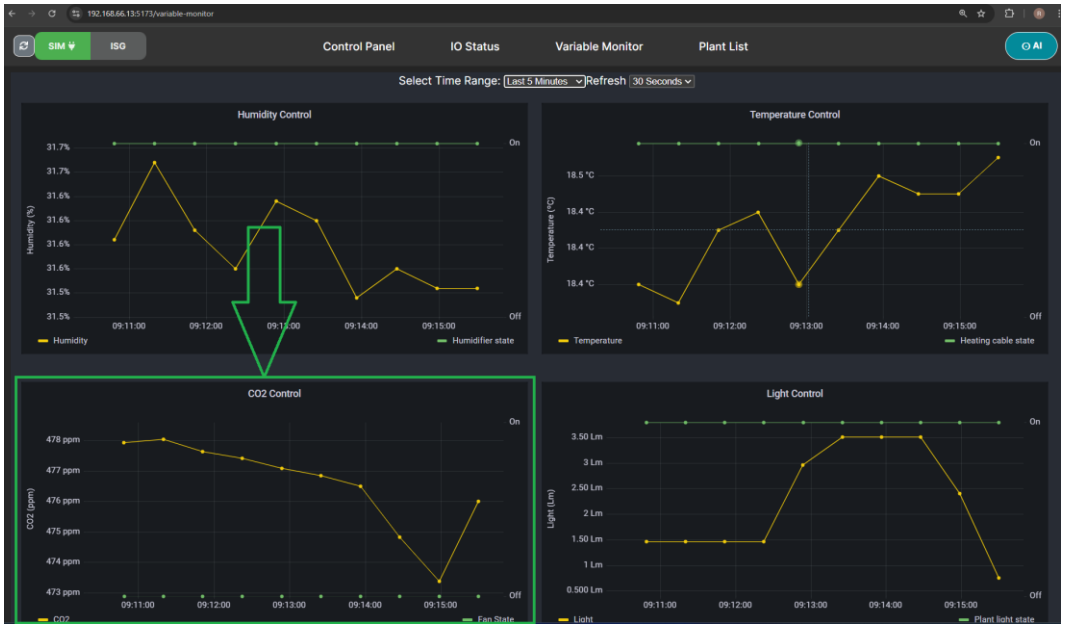

Ilustración 38: Gráfica con el valor de la concentración de CO2 y el estado del ventilador

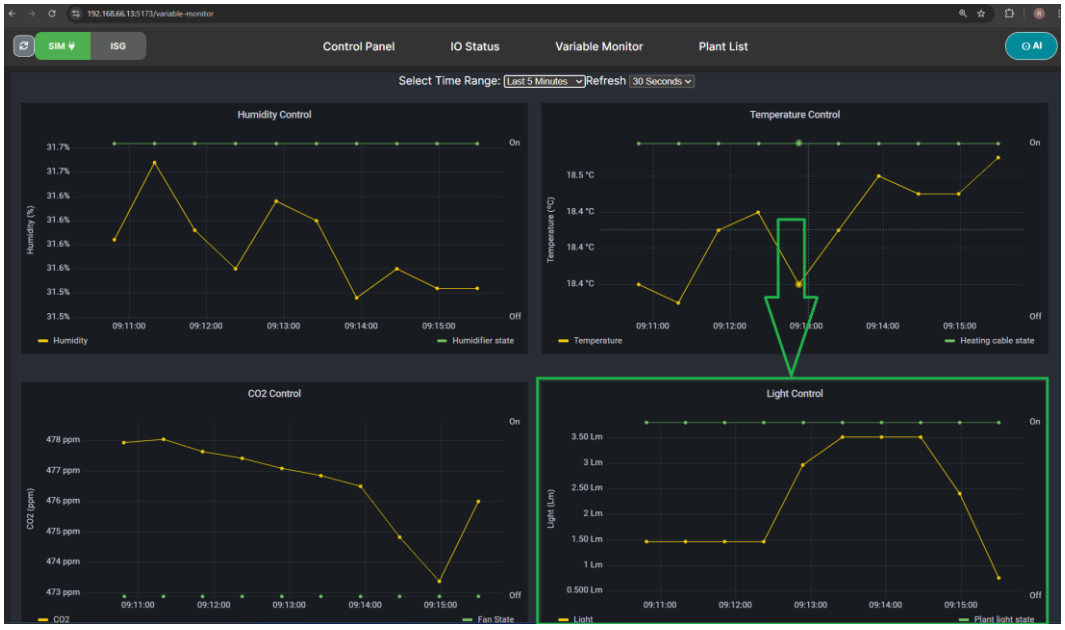

Ilustración 39: Gráfica con el valor de la luz y el estado de las luces LED

Debajo de los menús desplegables, encontrarás cuatro gráficos de Grafana. El gráfico superior izquierdo muestra el valor de la humedad en porcentaje relativo (%) y el estado del humidificador (Ilustración 36). El gráfico superior derecho muestra el valor de la temperatura en grados Celsius (°C) y el estado del cable calentador (Ilustración 37). El gráfico inferior izquierdo muestra el valor de concentración de CO2 en partes por millón y el estado del ventilador (Ilustración 38). El gráfico inferior derecho muestra el valor de luz en lux y el estado de las luces LED (Ilustración 39).

3.3.8 Panel Gestor de Plantas

En esta ventana (Ilustración 40), el usuario puede gestionar las plantas dentro del módulo y ver su estado actual.

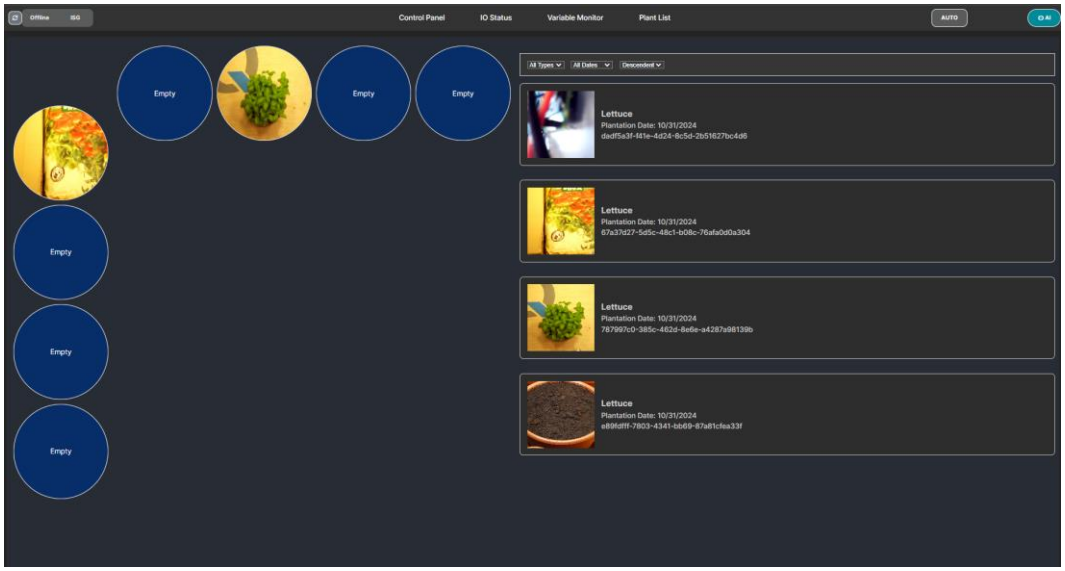

Ilustración 40: Ventana del panel gestor de plantas

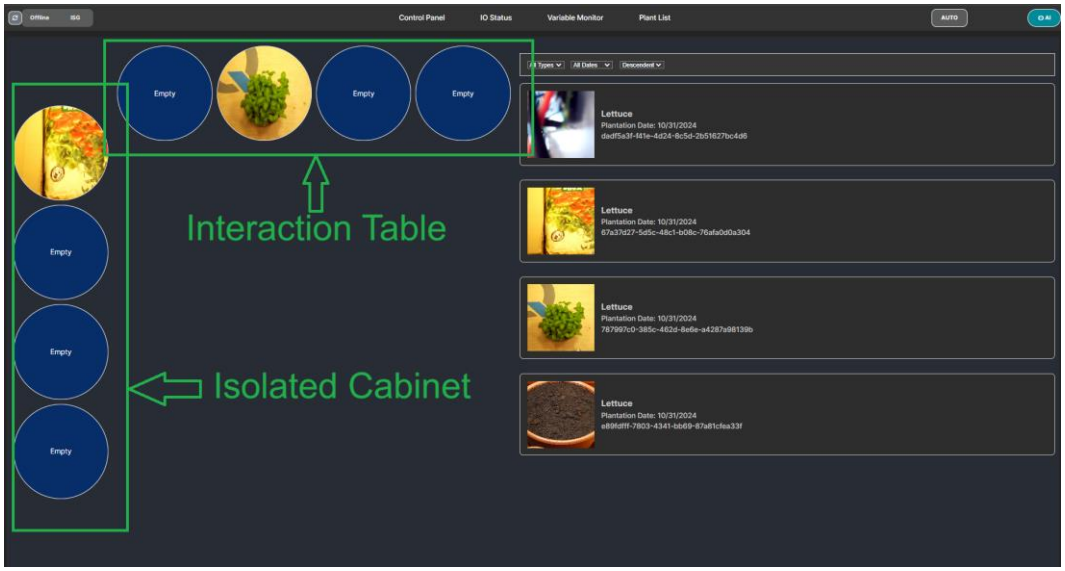

Ilustración 41: Tablero con la representación y disposición de las plantas del módulo

En la parte izquierda (Ilustración 41), hay un tablero que representa el diseño del módulo. En la parte superior (Señalado como 'Interaction Table' en la Ilustración 41), hay una fila con las plantas en la mesa de interacción. A la izquierda (Señalado como 'Isolated Cabinet' en la Ilustración 41), hay una columna con las plantas dentro del armario aislado. Cuando haces clic en una de las plantas, se desplegará un menú y puedes seleccionar diferentes acciones, dependiendo del estado actual de esa planta o maceta.

Cuando selecciones un espacio vacío en la mesa de interacción, aparecerá una ventana emergente que permite al usuario indicar que se ha colocado una maceta con tierra en ese lugar (Ilustración 42).

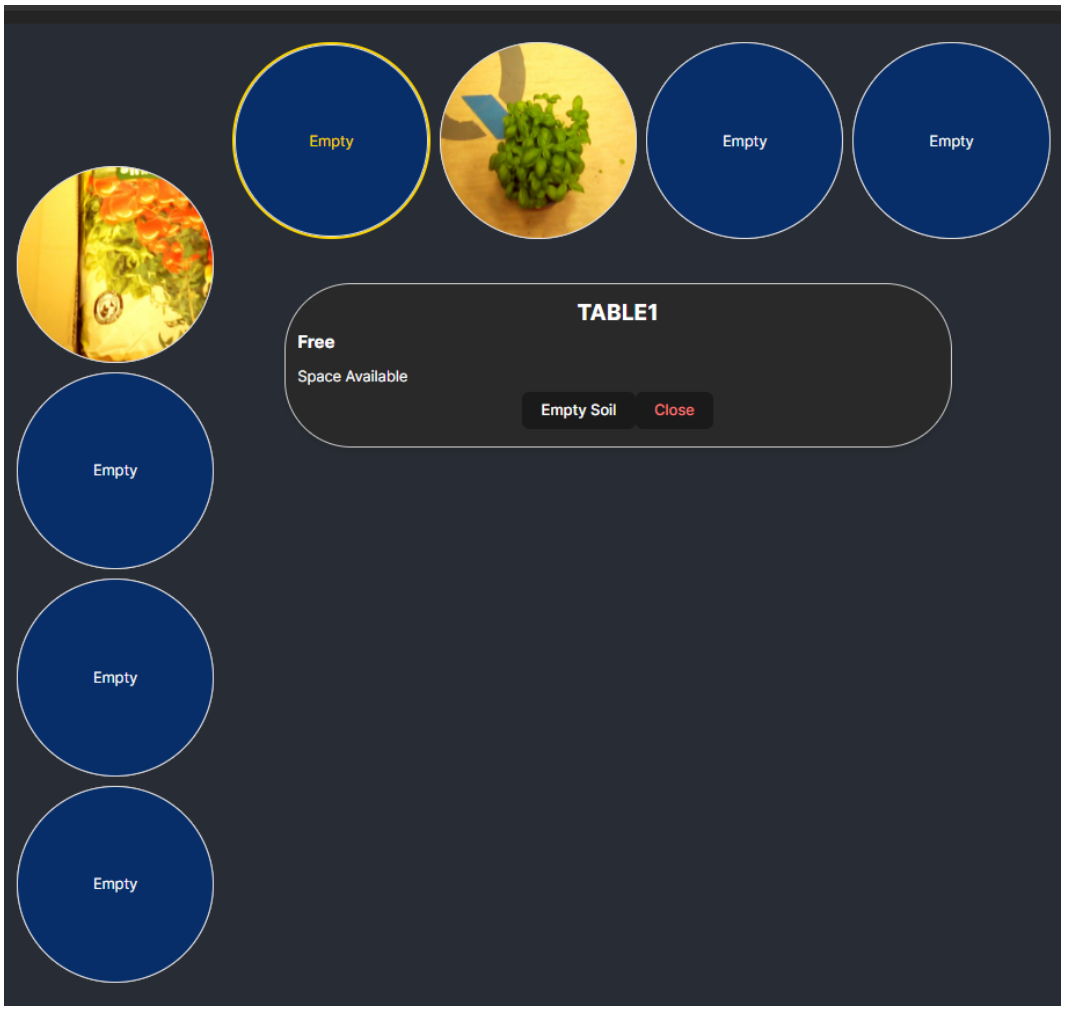

Ilustración 42: Ventana emergente para un espacio vacío en el gestor de plantas

Cuando pongas una maceta en un espacio vacío, la imagen del espacio cambiará para representar una maceta con tierra. Si seleccionas una maceta con tierra, aparecerá otra ventana emergente que te permite elegir si quieres plantar una semilla o eliminar la maceta del tablero (Ilustración 43).

Ilustración 43: Ventana emergente para una maceta con tierra

Después de seleccionar plantar una semilla en una maceta en la mesa de interacción, hacer clic en esa maceta con una semilla abrirá otra ventana emergente que te permite elegir si introducir la planta dentro del armario o eliminarla (Ilustración 44).

Ilustración 44: Ventana emergente para una maceta sembrada en la mesa de interacción

Si seleccionas una planta dentro del armario, aparecerá otra ventana emergente que te permite eliminar la planta del armario o tomar una foto y analizar su estado utilizando GPT (Ilustración 45).

Ilustración 45: Ventana emergente para una planta dentro del módulo aislado

En la parte derecha del panel, hay una lista de todas las plantas (Ilustración 46). Esta lista puede ordenarse por tipo de planta, fecha u orden (ascendente o descendente) utilizando tres menús desplegables. Cada entrada en la lista muestra la última foto de la planta, su tipo, la fecha de siembra y su número de identificación único.

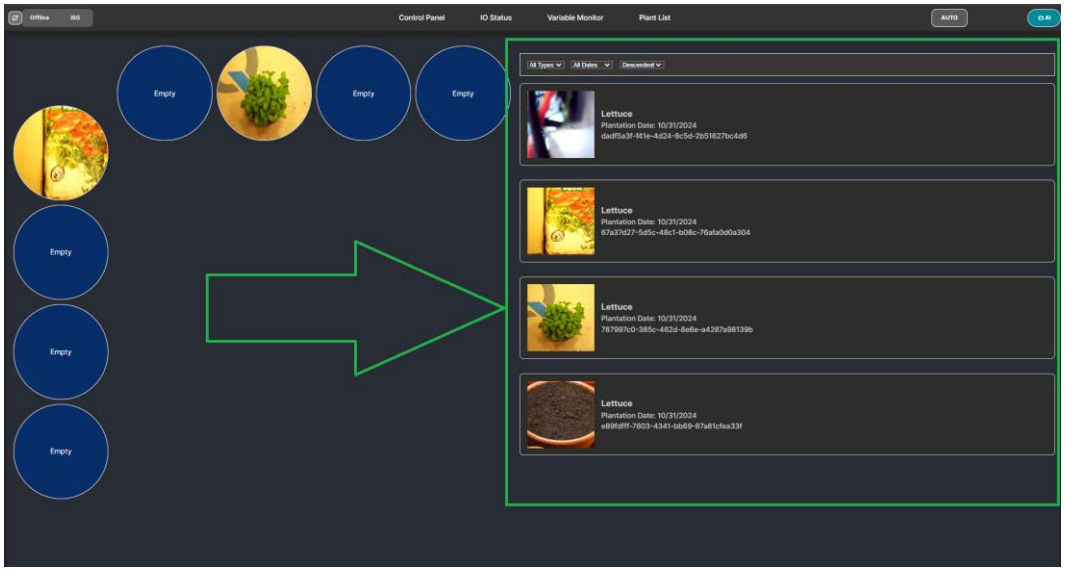

Ilustración 46: Lista del panel gestor de plantas

Si haces clic en una de las plantas de la lista, se desplegará un nuevo panel mostrando todas las fotos históricas junto con sus respectivos análisis de estado por parte de GPT (Ilustración 47).

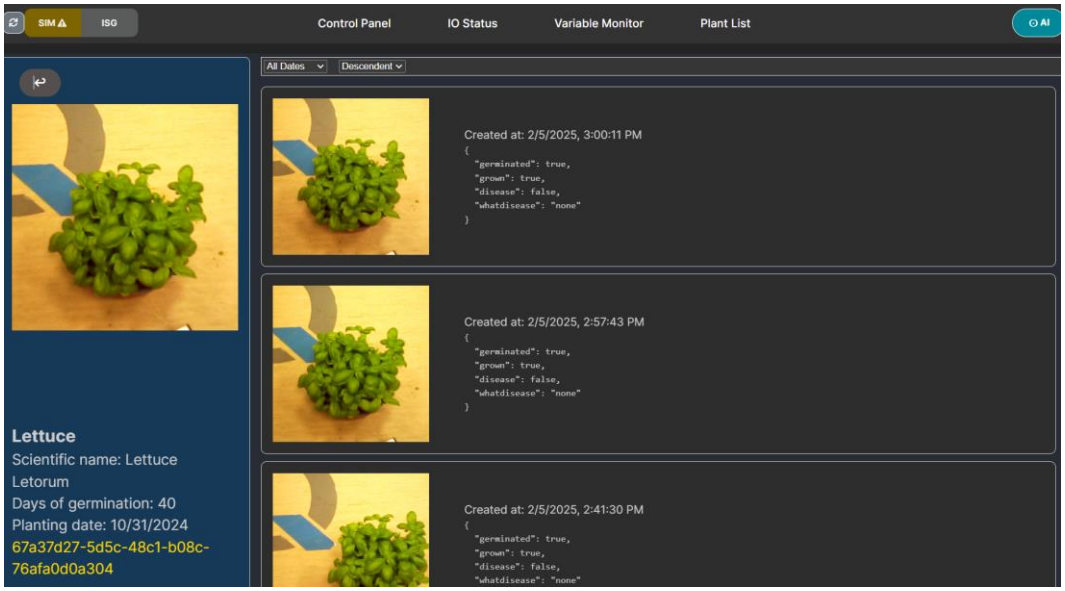

Ilustración 47: Panel con todas las fotos y el estado de la planta

4 Hardware: Selección del material, Diseño Eléctrico y Diseño de la Interfaz Raspberry Pi-Hardware

Este proyecto representa un desarrollo mecatrónico en el sentido más amplio, ya que integra disciplinas de ingeniería mecánica, eléctrica, electrónica, software y control.

Si bien el enfoque principal del proyecto es la implementación de métodos de vanguardia para el control de robots mediante software, es fundamental contar con un diseño de hardware sólido y bien estructurado. En cualquier sistema mecatrónico o ciberfísico, una arquitectura eléctrica robusta y específica para los objetivos del sistema es esencial para garantizar un funcionamiento óptimo. Un diseño eléctrico adecuado no solo facilita el desarrollo del software de control, sino que también contribuye a la estabilidad y eficiencia del sistema en su conjunto.

En este capítulo se detalla el trabajo relacionado con el hardware, incluyendo la selección de materiales esenciales para la construcción y operación del sistema, el diseño de los esquemas eléctricos, la implementación del armario de control eléctrico y el desarrollo de una interfaz de software que permite gestionar los elementos eléctricos a través de la Raspberry Pi.

4.1 Lista y Selección de Materiales

En este apartado se enumeran los elementos más importantes del proyecto. En el Anejo 1 puede verse la lista con todos los materiales utilizados en el proyecto, incluyendo una breve descripción y la dirección de compra.

Cada uno de los artículos seleccionados ha sido analizado en profundidad para confirmar la viabilidad de su uso y de su integración con el resto del sistema. Además, todos los artículos han sido comparados con otros similares y siempre se ha intentado la mejor opción, teniendo en cuenta sus características y su coste tanto directo como de funcionamiento.

Todos los enlaces han sido revisados por última vez en marzo de 2025.

4.1.1 Robot

- Robot Mitsubishi Melfa RV-7FC-D (Ilustración 48)
    - **Descripción:** Robot industrial utilizado para tareas automatizadas, como la recolección y siembra de semillas, el movimiento de macetas y la toma de fotos de las plantas.
    - **Precio:** Ya adquirido
    - **Enlace:** Mitsubishi Electric - Especificaciones del robot

Ilustración 48: Robot Mitsubishi Melfa RV-7FC-D

- Controlador CR750-D (Ilustración 49)
    - **Descripción:** Este controlador se utilizará para controlar el robot. Está programado en el lenguaje Melfa Basic-V y permite la comunicación mediante Ethernet.
    - **Precio:** Ya adquirido
    - **Enlace:** Manual del controlador CR750-D

Ilustración 49: Controlador CR750-D

- Montura Universal de Herramientas (UTM) preensamblada (Ilustración 50)
    - **Descripción:** Montura universal de herramientas para sujetar y detectar herramientas. A esta pieza se le ha rediseñado la

cubierta de plástico, que será impresa en 3D, al igual que se le ha añadido otra pieza para facilitar su acople con el robot Mitsubishi, el cableado eléctrico, de los tubos de regado y neumáticos; que permiten que cualquier herramienta disponga de ellas. También tiene un conjunto de conexiones eléctricas, que permiten leer o enviar señales desde la Raspberry Pi para controlar las herramientas.

- **Precio:** 60EUR
- **Enlace:** Farm.bot - UTM Preensamblada

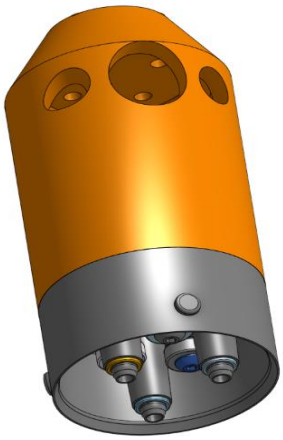

Ilustración 50: UTM preensamblada

- Herramientas diseñadas (Ilustración 51)
  - **Descripción:** Herramientas diseñadas para realizar las funciones de cuidado de las plantas, como recoger y sembrar semillas, sembrar o mover las macetas. Algunas de las herramientas necesitan ser cableadas para su correspondiente control eléctrico o necesitan un acople de un tubo de riego o neumático para cumplir sus funciones, Esto se realiza a través del UTM.
  - **Herramienta diseñada por nosotros**

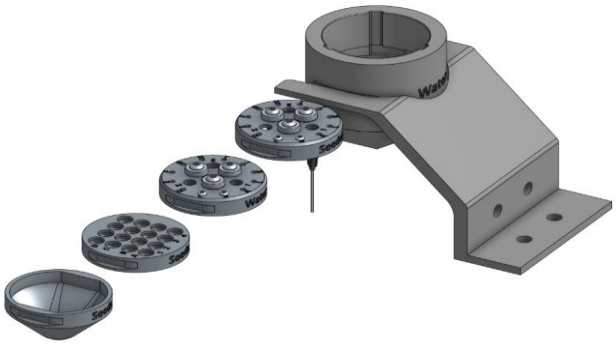

Ilustración 51: Herramientas diseñadas

4.1.2 Sensores

- SCD30 (Ilustración 52)
    - **Descripción:** Sensor de temperatura, humedad y CO2.
    - **Precio:** 44EUR
    - **Enlace:** Digi-Key - SCD30

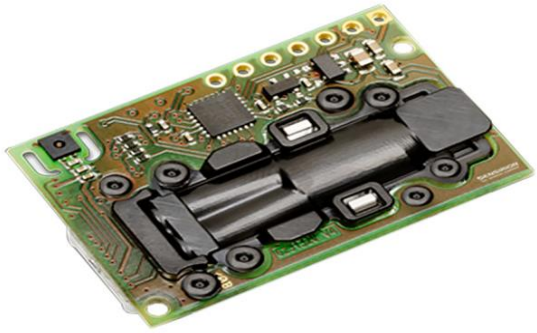

Ilustración 52: Sensor de temperatura, humedad y CO2

- TSL2591 (Ilustración 53)
    - **Descripción:** Sensor de luz.
    - **Precio:** 7.40EUR
    - **Enlace:** Digi-Key - TSL2591

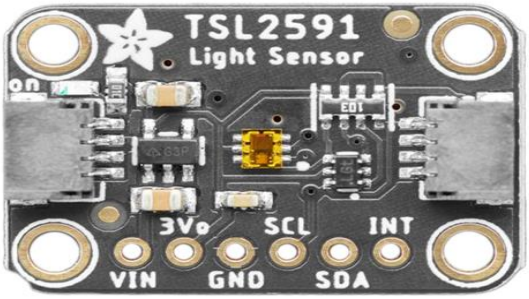

Ilustración 53: Sensor de luz

4.1.3 Actuadores

- Cable calefactor (Ilustración 54)
    - **Descripción:** Cable calefactor para reptiles que se colocará entre las macetas para calentar el módulo. (230VCA)
    - **Precio:** 25EUR
    - **Enlace:** Amazon - Cable calefactor

Ilustración 54: Cable calefactor

- Luces de espectro completo (Ilustración 55)
    - **Descripción:** Luces LED de espectro completo que proporcionarán a las plántulas el tipo de luz adecuado para su crecimiento. (230VCA)
    - **Precio:** 28EUR
    - **Enlace:** Amazon - Luces LED de espectro completo

Ilustración 55: Luces LED de espectro completo

- Electroválvula (riego) (Ilustración 56)
    - **Descripción:** Electroválvula para controlar el flujo de agua de riego. (24VCC)
    - **Precio:** 10EUR
    - **Enlace:** Amazon - Electroválvula

Ilustración 56: Electroválvula

- Ventilador (Ilustración 57)
    - **Descripción:** Ventilador para ventilar el módulo. Nos ayudará a renovar el aire y regular los niveles de CO_2 y humedad. (24VCC)
    - **Precio:** 6EUR
    - **Enlace:** Amazon - Ventilador 24V

Ilustración 57: Ventilador

- Humidificador (Ilustración 58)
    - **Descripción:** Humidificador ultrasónico que nos ayudará a regular los niveles de humedad relativa del aire. (Cable USB)
    - **Precio:** 6EUR
    - **Enlace:** Amazon - Humidificador ultrasónico

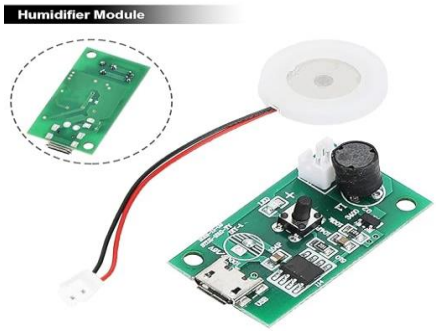

Ilustración 58: Humidificador

- Motor paso a paso Nema 17 con caja reductora planetaria de relación 50:1 MG Series (Ilustración 59):
    - **Descripción:** Motor paso a paso para controlar el movimiento de la puerta deslizante del módulo.
    - **Precio:** 34EUR
    - **Enlace:** Stepper Online - Motor Nema 17

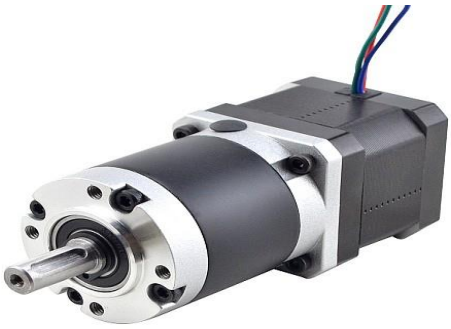

Ilustración 59: Motor paso a paso

- Bomba de vacío de 24V con filtro EMI (Ilustración 60):
    - **Descripción:** Bomba de vacío para recoger semillas por absorción.
    - **Precio:** 40EUR
    - **Enlace:** Farm.bot - Bomba de vacío

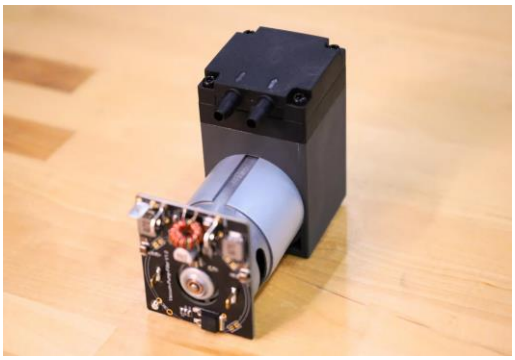

Ilustración 60: Bomba de vacío

4.1.4 Dispositivos del Armario Eléctrico

- Relé de 24V (Ilustración 61)
    - **Descripción:** Módulo de relés de 8 canales para 24VCC y lógica de control de 3.3V-5V.
    - **Precio:** 12EUR
    - **Enlace:** Amazon - Módulo de relés 24V

Ilustración 61: Relé de 24VDC

- Relé de 220V (Ilustración 62)
    - **Descripción:** x5 Módulo de relé de 1 canal para 230VCA y lógica de control de 3.3V.
    - **Precio:** 11EUR
    - **Enlace:** Amazon - Relé de 230V

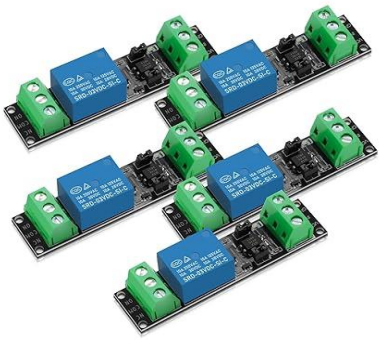

Ilustración 62: Relé de 230VAC

- Fuente de alimentación SITOP PSU100S, 24VCC 5A 144W (Ilustración 63)
    - **Descripción:** Dispositivo de alimentación para transformar 230VCA a 24VCC.
    - **Precio:** Gratis (ya adquirido)
    - **Enlace:** RS Components - Fuente de alimentación

Ilustración 63: Fuente de alimentación de 24VDC

- Fusible Seccionador Automático ABB ST201M-B10 (Ilustración 64)
    - **Descripción:** Fusible de protección eléctrica de 10A.
    - **Precio:** Gratis (ya adquirido)
    - **Enlace:** RS Components - Interruptor automático

Ilustración 64: Fusible seccionador

- Relé de seguridad Pilz PNOZ s5 (Ilustración 65)
    - **Descripción:** Relé de seguridad para cortar la corriente a ciertos actuadores de maniobra en caso de parada de emergencia.
    - **Precio:** Gratis (ya adquirido)
    - **Enlace:** Pilz - Relé de seguridad

Ilustración 65: Relé de seguridad

4.1.5 Controlador

- Raspberry Pi 4B (4GB) (Ilustración 66)
    - **Descripción:** Pequeño ordenador de placa única con procesador ARM, doble HDMI y USB 3.0. Soporta sistemas operativos basados en Linux y se utiliza para programación y automatización. Es el núcleo lógico y controlador de nuestro sistema.
    - **Precio:** Gratis (ya disponible)
    - **Enlace:** DigiKey - Raspberry Pi 4B

Ilustración 66: Raspberry Pi 4B

4.2 Diseño y Análisis Eléctrico

Dado que en el sistema mecatrónico diseñado hay un gran número de dispositivos y actuadores eléctricos, es indispensable realizar los análisis, cálculos y diseños eléctricos pertinentes antes de implementar los distintos elementos eléctricos. Para ello, la mejor practica es construir un centro de control eléctrico, donde se controlen tanto la lógica de control como la potencia eléctrica. Por lo tanto, se ha diseñado y construido un armario eléctrico.

Este apartado cubre la parte eléctrica del proyecto, incluyendo los códigos y estándares eléctricos seguidos, los análisis y cálculos realizados, los esquemas eléctricos y el diseño del armario eléctrico.

4.2.1 Código de Colores para Cables:

El código de colores seguido para el cableado eléctrico es el establecido en el estándar **[10]**, tal y como se muestra a continuación:

Tierra o Cable de Protección Eléctrica

- Tierra: Verde y Amarillo.

Monofásico

- Fase: Marrón
- Neutro: Azul claro

Trifásico

- L1: Marrón
- L2: Negro
- L3: Gris
- N: Azul claro

24VDC:

- +24V: Azul oscuro

- -0V: Blanco

4.2.2 Código de Colores para los Terminales:

El código de colores para los borneros del armario eléctrico se ha realizado sin seguir una norma específica, ya que no existe un código de colores establecido para los borneros.

Gris

Línea

Azul

Neutro

Rojo

Corriente continua positiva (24V, 5V y 3.3V)

Blanco

0VDC

Negro

Señales de control y entradas y salidas digitales.

4.2.3 Cálculos Eléctricos

Comparación de Estimaciones del Fabricante y Datos de Prueba

En la Tabla 3 se muestra una comparación entre los datos estimados por el fabricante y los datos obtenidos a través de pruebas para los actuadores eléctricos.

Tabla 3: Tabla de la comparación de estimaciones del fabricante y los datos obtenidos en pruebas

Componente	Voltaje (V)	Corriente Teórica (A)	Corriente Medida (A)
Cable Calefactor	230VAC	0.065A	0.05A
Luces LED	230VAC	0.1956A	0.33A
Ventilador	24VDC	0.11A	0.1A
Humidificador	230VAC	0.009A	0A (No se detectó corriente)
Electroválvula	24VDC	N/A	0.22A
Bomba de Vacío	24VDC	N/A	0.45A
Cerradura	24VDC	0.57A	0.54A

Motor Paso a Paso	24VDC	1.58A	1.61A
Robot y Controlador	230VAC	N/A	3.5A
Raspberry Pi	230VAC	N/A	1A

Las corrientes consumidas por los sensores; 20mA, según la hoja de características del fabricante; se consideran despreciables para estos cálculos.

Consumo del Robot:

Nota: Como el fabricante no proporciona especificaciones eléctricas, realizamos pruebas para determinar la corriente nominal y de pico del robot en condiciones de carga superiores a las condiciones normales de funcionamiento, los resultados de las pruebas están recogidos en la tabla 4.

Tabla 4: Consumos de corriente del robot

Estado del Robot	Consumo de Corriente (A)
En reposo	0.48 A
Servos activos pero estáticos	0.87 A
En movimiento	1.4 A
Activación de servos	3.5 A

La corriente de pico se presenta cuando activamos los servos después de un largo período de inactividad. Esto se debe a la corriente de irrupción necesaria para magnetizar los motores.

Resumen Total de Corriente

En la tabla 5 se muestran los consumos para 24VDC, 230VAC y el total.

Tabla 5: Tabla con los consumos de corriente continua, alterna y total.

Tipo de Voltaje	Corriente Total (A)
24V DC	2.92A
230V AC	4.88A
Total	**7.8A**

4.2.4 Selección de Cables

Introducción

El objetivo es proporcionar una justificación técnica para la selección del calibre de los cables eléctricos en un armario eléctrico industrial, siguiendo la norma **[11]**. El proceso de selección considera múltiples factores, tales como la capacidad de corriente (ampacidad), caída de tensión, protección contra cortocircuitos y durabilidad mecánica.

Factores Considerados en la Selección del Tamaño del Cable

Ampacidad (Capacidad de Conducción de Corriente)

La norma **[11]** proporciona directrices sobre la capacidad máxima de corriente que un conductor puede manejar sin exceder los límites de temperatura. Dado que la instalación consiste en conductores de cobre en un cable de tres hilos (fase, neutro, tierra) fijado a una pared o suelo, se consideran los siguientes valores de ampacidad:

Sección del Cable (mm^2)	Corriente Máxima (A)
0.5 mm^2	5.0 A
0.75 mm^2	6.0 A
1.0 mm^2	10.0 A
1.5 mm^2	16.0 A
2.5 mm^2	20.0 A
4.0 mm^2	25.0 A
6.0 mm^2	32.0 A
10.0 mm^2	40.0 A

La corriente total tomada de la fuente de alimentación principal es de 7.8A, lo que supera el límite seguro de 0.75 mm^2, pero está bien dentro del límite de 1.0 mm^2.

Cálculo de Caída de Tensión

La caída de tensión debe limitarse al 4% (9.2V) para circuitos de 230V AC y al 5% (1.2V) para circuitos de 24V DC, con el fin de garantizar el correcto funcionamiento de los equipos.

La fórmula general de la caída de tensión es:

$$\Delta V = I \times L \times R$$

donde:

- (I) = corriente (A)
- (L) = longitud total del cable (m, ida y vuelta)
- (R) = resistencia del conductor (Ω/m)

Caída de Tensión para 230V AC (Alimentación Principal)

Se realizarán los cálculos para los cables de alimentación que dan corriente a todo el armario eléctrico:

Para cable de 1.0 mm^2:

- Resistencia: 19 Ω/km
- Corriente: 7.8A
- Longitud: 10m (5m de ida)

$$\Delta V = 7.8A \times 10m \times 0.019\Omega/m = 1.48V$$

$$\frac{1.48V}{230V} \times 100 = 0.64\%$$

- **Conclusión**: La caída de tensión está muy por debajo del límite del 4%, lo que confirma que el cable de 1.0 mm² es adecuado para las líneas de alimentación principales.

Caída de Tensión para 24V DC (Actuadores y Sensores)

Para cable de 0.75 mm²:

- Resistencia: 26 Ω/km
- Corriente: 2.92A
- Longitud: 10m

$$\Delta V = 2.92A \times 10m \times 0.026\Omega/m = 0.76V$$

$$\frac{0.76V}{24V} \times 100 = 3.16\%$$

- **Conclusión**: La caída de tensión está dentro del límite del 5%, lo que confirma que el cable de 0.75 mm² es suficiente para los circuitos de 24V DC.

Protección contra Cortocircuitos

El cable debe soportar corrientes de cortocircuito sin dañarse. La fórmula aportada en [11] para la resistencia térmica es:

$$I^2 \times t \leq k^2 \times S^2$$

donde:

- (I) = corriente de cortocircuito (A)
- (t) = tiempo de desconexión (s)
- (k) = constante del material para cobre (143 para aislamiento de PVC)
- (S) = tamaño del conductor (mm²)

Para disyuntores de ≤10A, los cables de 1.0 mm² y 0.75 mm²

Durabilidad Mecánica

- Se elige **1.0 mm²** para los cables de alimentación principal debido a su mayor resistencia mecánica y su capacidad para soportar esfuerzos de manipulación.
- **0.75 mm²** es aceptable para el cableado interno, ofreciendo suficiente durabilidad y flexibilidad.

Selección Final de Cables

En la Tabla 6 se muestra la selección final de cables.

Tabla 6: Selección final de cables

Aplicación	Voltaje	Corriente (A)	Tamaño del Cable
Alimentación Principal	230V AC	7.8A	**1.0 mm² (3x1)**
Luces LED, Cable Calefactor	230V AC	≤ 0.33A	**0.75 mm²**
Robot y Controlador	230V AC	3.5A (pico)	**0.75 mm²**
Actuadores 24V (Ventilador, Bomba, Válvula)	24V DC	≤ 2.92A	**0.75 mm² (2x0.75)**
Sensores	3.3-5V DC	N/A	**0.5 mm² (Apantallado)**

Para los sensores se necesitan 4 cables. Como los sensores operan a 3.3-5VDC y su consumo de corriente es despreciable, utilizaremos cables de **0.5 mm²**. Para lograr un cableado más limpio y organizado, los cables estarán dentro de una **manguera multihilo**. Además, para evitar interferencias electromagnéticas (EMI), dado que estos cables transportan datos sensibles, la manguera de cables debe ser apantallada.

> **Nota**: Como las mangueras **2x0.75 mm²** son raras y difíciles de adquirir, usaremos **2x1 mm²**, ya que este cambio no afecta negativamente el rendimiento del sistema.

- **Conclusión**: Esta selección garantiza una operación segura, cumplimiento con la norma **[11]**, y optimiza el uso de materiales manteniendo la fiabilidad del sistema.

4.3 Esquemas Eléctricos y de la Disposición del Armario Eléctrico

En el anejo 2 se pueden observar los esquemas eléctricos y de la disposición del armario eléctrico junto con las imágenes del estado actual de su implementación, que está avanzada pero inconclusa. El armario eléctrico utilizado ha sido reciclado, ya que fue sobrante de otro proyecto, pero todo el diseño de la disposición y eléctrico es propio.

4.4 Implementación de la Biblioteca de Control de Sensores y Actuadores para Raspberry Pi

Una vez explicado cómo se ha seleccionado el material y como se ha diseñado el control eléctrico, procederemos a explicar cómo se ha desarrollado el

software para construir una interfaz entre el controlador, una Raspberry Pi 4B, y los sensores, actuadores y dispositivos eléctricos presentados en los anteriores apartados.

Este apartado proporciona una guía completa sobre la estructura, uso e integración con el servidor de una biblioteca para Raspberry Pi, diseñada para controlar los sensores y actuadores utilizados en este proyecto. La biblioteca permitirá la gestión de actuadores de CC y CA, así como sensores ambientales y otras acciones lógicas necesarias, a través de GPIO e I2C. Además, se integra con el backend FastAPI para permitir monitoreo en tiempo real, gestión de umbrales y transmisión de datos mediante conexiones WebSocket.

4.4.1 Descripción General

La biblioteca facilita el control de varios tipos de actuadores y la lectura de datos de sensores conectados a una Raspberry Pi. Diseñada para aplicaciones de monitoreo ambiental, gestiona los actuadores en función de valores umbral y transmite datos en tiempo real a un servidor FastAPI.

La biblioteca se divide en dos módulos principales:

1. **Módulo de Actuadores**: Gestiona el control basado en GPIO para un ventilador, solenoide, humidificador, cable calefactor, luces, un motor paso a paso y el reinicio de emergencia.
2. **Módulo de Sensores**: Proporciona funciones para inicializar y leer datos de sensores de CO_2, temperatura, humedad, luz, dos interruptores magnéticos y el estado de emergencia.

Adicionalmente, la **clase IOController** (en plant_manager.py) coordina estas operaciones administrando valores umbral y recorriendo las lecturas de los sensores para controlar los actuadores. Esta clase se integra con el servidor FastAPI definido en main.py, proporcionando endpoints de API REST y conexiones WebSocket para monitoreo de datos en tiempo real.

4.4.2 Módulos de la Biblioteca

Módulo de Actuadores

El módulo actuators.py proporciona funciones de control para actuadores basados en GPIO.

Asignación de Pines para Actuadores

En la Tabla 7 se muestra la asignación de pines de la Raspberry Pi para los actuadores.

Tabla 7: Asignación de pines de la Raspberry Pi para los actuadores.

Actuador	Número de Pin	Voltaje	Nivel Lógico para Activar
Ventilador	27	24V DC	BAJO
Solenoide	22	24V DC	BAJO
Bomba de vacío	20	24V DC	BAJO
Cerradura	24	24V DC	BAJO
Humidificador	23	230V AC	ALTO
Cable Calefactor	17	230V AC	ALTO
Luces	25	230V AC	ALTO
Motor Paso a Paso ENA+	19	3.3V (control)	ALTO
Motor Paso a Paso DIR+	16	3.3V (control)	ALTO
Motor Paso a Paso PUL+	13	3.3V (control)	ALTO
Reinicio de Emergencia	8	Detección de continuidad	BAJO

Inicialización

La configuración de GPIO se inicializa en modo BCM, y cada pin se configura como una salida. El estado predeterminado de cada actuador se establece en "apagado".

Funciones

- **Control del Ventilador**
  - activate_fan(): Activa el ventilador configurando su pin GPIO en BAJO.
  - deactivate_fan(): Desactiva el ventilador configurando su pin GPIO en ALTO.
  - is_fan_active() -> bool: Devuelve True si el ventilador está activo (BAJO), de lo contrario, False.
- **Control del Solenoide**
  - activate_solenoid(): Activa el solenoide configurando su pin GPIO en BAJO.
  - deactivate_solenoid(): Desactiva el solenoide configurando su pin GPIO en ALTO.
  - is_solenoid_active() -> bool: Devuelve True si el solenoide está activo (BAJO), de lo contrario, False.

- Control del Humidificador
    - activate_humidifier(): Activa el humidificador configurando su pin GPIO en ALTO.
    - deactivate_humidifier(): Desactiva el humidificador configurando su pin GPIO en BAJO.
    - is_humidifier_active() -> bool: Devuelve True si el humidificador está activo (ALTO), de lo contrario, False.
- Control del Cable Calefactor
    - activate_heating_cable(): Activa el cable calefactor configurando su pin GPIO en ALTO.
    - deactivate_heating_cable(): Desactiva el cable calefactor configurando su pin GPIO en BAJO.
    - is_heating_cable_active() -> bool: Devuelve True si el cable calefactor está activo (ALTO), de lo contrario, False.
- Control de las Luces
    - activate_lights(): Activa las luces configurando su pin GPIO en ALTO.
    - deactivate_lights(): Desactiva las luces configurando su pin GPIO en BAJO.
    - is_lights_active() -> bool: Devuelve True si las luces están activas (ALTO), de lo contrario, False.
- Control del Cerrojo
    - activate_locker(): Activa el cerrojo configurando su pin GPIO en BAJO.
    - deactivate_locker(): Desactiva el cerrojo configurando su pin GPIO en ALTO.
    - is_locker_active() -> bool: Devuelve True si el cerrojo está activo (BAJO), de lo contrario, False.
- Emergency Reset Control
    - reste_emergency(): Activa un relé que cierra la continuidad entre los pines 34 y 12 del relé de seguridad, reiniciando así el relé de emergencia.

Módulo de Sensores

El módulo sensors.py proporciona funcionalidad para inicializar y leer datos de los sensores:

- **SCD30** para la concentración de CO_2, temperatura y humedad.

- **TSL2591** para la intensidad de luz (lux).
- **UTM_C** para reconocer si hay una herramienta montada.
- **Interruptores magnéticos reed** para detectar si la puerta automatizada está completamente abierta o cerrada.

Asignación de Pines para Sensores

En la Tabla 8 se muestra la asignación de pines de la Raspberry Pi para los sensores.

Tabla 8: Asignación de pines de la Raspberry Pi para los sensores.

Sensor	Número de Pin	Nivel Lógico para Activar
SCD30	I2C	N/A
TSL2591	I2C	N/A
UTM_C	21	BAJO (Pull Up)
MRS-superior	6	BAJO (Pull Up)
MRS-inferior	5	BAJO (Pull Up)
Detección de emergencia	7	BAJO (Pull Up)

Funciones

- **Sensor SCD30 (CO_2, Temperatura, Humedad)**
  - initialize_scd30(i2c): Inicializa el sensor SCD30 en el bus I2C.
  - read_scd30_data(scd30): Lee los datos de CO_2, temperatura y humedad desde el sensor SCD30.
- **Sensor TSL2591 (Intensidad de Luz)**
  - initialize_tsl2591(i2c): Inicializa el sensor TSL2591 en el bus I2C.
  - read_tsl2591_lux(tsl2591): Lee la intensidad de luz en lux desde el sensor TSL2591.
- **Pin UTM_C**
  - initialize_utm_c(utm_c_pin=21): Inicializa el pin UTM_C como una entrada con una resistencia pull-up.
  - is_tool_mounted(utm_c_pin = 21): Lee el valor del pin digital especificado o el 21 por defecto.
- **Interruptores Magnéticos Reed Switches**
  - initialize_reed_switch_up(pin=6): Inicializa un sensor magnético de interruptor de láminas para detectar si la puerta está totalmente levantada configurando el pin GPIO 6 como una entrada con una resistencia de pull-up.

- initialize_reed_switch_down(pin=5): Inicializa un sensor de interruptor de láminas magnético para detectar si la puerta está totalmente cerrada configurando el pin GPIO 5 como una entrada con una resistencia de pull-up.
- is_sliding_door_up(): Comprueba si la puerta corredera está en posición elevada.
- is_sliding_door_dow(): Comprueba si la puerta corredera está en posición bajada.

- **Detección de la parada de emergencia**
  - Para detectar la emergencia configuraremos el pin 7 para producir una interrupción e iniciar la función encargada de manejar las paradas de emergencia.

```
E_STOP_PIN = 7
GPIO.setup(E_STOP_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)
## Trigger callback on emergency stop activation (e.g. rising edge if no
rmally-closed)
GPIO.add_event_detect(E_STOP_PIN, GPIO.RISING, callback=self._han
dle_emergency_stop, bouncetime=200)
```

4.4.3 Integración con el Servidor IO

Clase IOController en io_controller.py

La clase IOController, declarada en io_controller.py, actúa como el coordinador principal para gestionar las operaciones de sensores y actuadores basándose en los umbrales establecidos. Administra la actualización de umbrales, lee los datos de los sensores y controla los actuadores en consecuencia.

Funciones Clave en IOController

- **prepare_io()**: Inicializa las conexiones GPIO e I2C para los sensores y configura los pines GPIO de cada actuador.
- **update_thresholds()**: Escucha de manera asíncrona las actualizaciones de umbrales desde una cola, permitiendo ajustes dinámicos de los mismos.
- **io_control_loop()**: Bucle de control principal que lee datos de los sensores, los compara con los umbrales y activa o desactiva los actuadores en consecuencia. Además, mantiene un promedio de los datos de los sensores para su registro en la base de datos.

Umbrales y Control de Actuadores

El IOController ajusta los siguientes actuadores según los umbrales de los sensores:

- **Ventilador**: Se activa cuando el CO_2 supera el umbral establecido.

- **Cable Calefactor**: Se activa cuando la temperatura cae por debajo del umbral establecido.
- **Humidificador**: Se activa cuando la humedad cae por debajo del umbral establecido.
- **Luces**: Se activan cuando la intensidad de la luz cae por debajo del umbral establecido.

Desglose Detallado del Bucle de Control IO

El método io_control_loop() en IOController es responsable de manejar las operaciones principales de IO y el control de actuadores basado en los datos de los sensores, con un enfoque en mantener un entorno estable y reducir la activación y desactivación rápida de los actuadores. A continuación, se describe su funcionamiento y lógica:

1. **Lectura y Validación de Sensores**:
    - Lee los datos del sensor SCD30 (CO_2, temperatura, humedad) y del sensor TSL2591 (intensidad de luz).
    - Valida las lecturas para asegurarse de que los datos están disponibles antes de almacenarlos o utilizarlos. Esto es crucial para evitar que datos inválidos afecten las decisiones de activación de los actuadores.
2. **Control de Actuadores con Histéresis**:
    - Se implementa histéresis para evitar que los actuadores se enciendan y apaguen con frecuencia debido a pequeñas fluctuaciones alrededor de los valores umbral. Por ejemplo:
        - El ventilador se activa cuando el CO_2 supera el umbral y solo se desactiva cuando el CO_2 cae por debajo del umbral menos un valor de margen (por ejemplo, CO2_THRESHOLD - 75). Este margen reduce la activación innecesaria del actuador, prolongando la vida útil del equipo y manteniendo la estabilidad.
    - De manera similar, el cable calefactor y el humidificador se activan y desactivan dentro de un pequeño margen alrededor de sus respectivos umbrales.
3. **Mantenimiento de Promedios de Lecturas**:
    - El bucle mantiene listas de lecturas recientes (hasta 15 ciclos) para CO_2, temperatura, humedad e intensidad de luz.
    - Cada 15 ciclos, se calcula el promedio de cada parámetro, que luego se guarda en la base de datos. Este promedio suaviza fluctuaciones transitorias y proporciona una representación más

precisa de las condiciones ambientales para el análisis a largo plazo.

4. **Detección de Cambios de Estado y Actualización de la Base de Datos**:
    - Se realiza un seguimiento de los estados previos de cada actuador.
    - Cuando un actuador cambia de estado (por ejemplo, el ventilador se enciende o apaga), se actualiza la base de datos con este nuevo estado. Este enfoque garantiza que solo los cambios reales de estado desencadenen escrituras en la base de datos, minimizando entradas redundantes.
5. **Transmisión de Datos en Tiempo Real a WebSocket**:
    - En cada ciclo, los últimos estados de los sensores y actuadores se colocan en una cola de datos. Esta cola es consumida por el endpoint WebSocket /ws-io, que transmite datos a los clientes conectados para el monitoreo en tiempo real.

Endpoints de la API del Servidor y WebSockets en main.py

El servidor FastAPI en main.py gestiona las conexiones WebSocket y los endpoints de la API REST para interactuar con IOController. La funcionalidad clave del servidor incluye la transmisión de datos en tiempo real a través de WebSockets y la gestión de umbrales mediante endpoints REST.

WebSocket para Datos IO en Tiempo Real (/ws-io)

El endpoint WebSocket /ws-io transmite datos IO en tiempo real desde sensor_data_queue, permitiendo a los clientes monitorear continuamente las lecturas de sensores y los estados de los actuadores.

```
@app.websocket("/ws-io")
async def websocket_io_endpoint(websocket: WebSocket):
    await websocket.accept()
    is_connected = True
    try:
        while not shutdown_event.is_set():
            io_state = await sensor_data_queue.get()  # Obtener los últimos datos del sensor

            await websocket.send_json(io_state)  # Enviar los datos del sensor en formato JSON
    except Exception as e:
        logging.error("WebSocket de IO cerrado:", e)
        is_connected = False
    finally:
```

```
        if is_connected:
            await websocket.close()
```

Endpoint de la API para Configurar Umbrales de Sensores (/set-thresholds)

El endpoint /set-thresholds permite actualizar los valores de umbral utilizados por IOController para gestionar la activación de actuadores según los datos de los sensores.

- **Ruta**: /set-thresholds
- **Método**: POST
- **Parámetros**:
    - CO2: Umbral de CO_2 en ppm.
    - temperature: Umbral de temperatura en grados Celsius.
    - humidity: Umbral de humedad en porcentaje.
    - lux: Umbral de intensidad de luz en lux.
- **Respuesta**: Confirma los valores de umbral actualizados.

```
@app.post("/set-thresholds")
async def set_thresholds(thresholds: dict):
    try:
        thresholds_converted = {
            "CO2": float(thresholds.get("CO2", io_controller.CO2_THRESHOLD)),
            "temperature": float(thresholds.get("temperature", io_controller.TEMPE
RATURE_THRESHOLD)),
            "humidity": float(thresholds.get("humidity", io_controller.HUMIDITY_THR
ESHOLD)),
            "lux": float(thresholds.get("lux", io_controller.LUX_THRESHOLD)),
        }
        await io_controller.threshold_update_queue.put(thresholds_converted)
        return {"status": "Umbrales actualizados con éxito", "thresholds": threshol
ds_converted}
    except ValueError as e:
        raise HTTPException(status_code=400, detail=f"Valor de umbral inválido:
{e}")
    except Exception as e:
        raise HTTPException(status_code=500, detail=f"Error al actualizar los um
brales: {e}")
```

Apagado y Limpieza

El servidor en main.py realiza un apagado controlado, incluyendo la limpieza de los recursos GPIO y la señalización de terminación para todas las tareas y conexiones WebSocket, asegurando que todos los recursos sean liberados correctamente.

```
await io_controller.prepare_io()  # Preparar configuraciones y ajustes de IO
# Procedimientos de apagado
io_controller.threshold_update_queue.put("shutdown")
shutdown_event.set()
```

5 Robot Mitsubishi Melfa RV-7FC-D: Programación, Comunicación y Gemelo Digital

Para el desarrollo de este proyecto era necesario el uso de un robot para automatizar las tareas de manipulación y plantado de las plantas y se nos dio la oportunidad de utilizar un robot Mitsubishi Melfa RV-7FC-D y su controlador CR750-D, que estaba disponible en la empresa ya que no fue utilizado para el proyecto que estaba previsto, pero que no disponía de la licencia del software oficial que proporciona Mitsubishi para su control ya que fue utilizada para el proyecto mencionado.

Por lo tanto, una de nuestras primeras tareas en este proyecto fue investigar si era posible programar, controlar y comunicarse con el robot sin el uso del software oficial. Decidimos invertir tiempo y trabajo en averiguar la viabilidad del uso de este robot para nuestros objetivos, y tras leer las documentaciones oficiales del robot [12] [13] y de su controlador [14] [15] [16], llegamos a la conclusión de que sería posible si desarrollábamos un protocolo de comunicación que permitiese el control del robot por parte de una Raspberry Pi.

Tras la fase de investigación tomamos la decisión de utilizar este robot y una Raspberry Pi para controlar el sistema, por lo tanto, desarrollamos el protocolo de comunicación descrito en profundidad en este capítulo, aprovechando además las capacidades multitarea del controlador del robot.

Mientras desarrollábamos dicho protocolo de comunicación, nos encontramos con la dificultad de desarrollar programas para el robot sin la licencia del software proporcionado por Mitsubishi, ya que no disponíamos de ningún editor de código ni compilador. Es por ello por lo que desarrollamos nuestras propias herramientas para desarrollar el software del robot, que analizaremos en este capítulo más adelante.

Adicionalmente, para la empresa era sumamente importante el desarrollo de un Gemelo Digital, ya que no solo es útil para poder realizar pruebas y simulaciones sin peligro antes de probar los programas en el robot real, sino que también permite que un trabajador en cualquier parte del mundo pueda trabajar y contribuir en el proyecto, algo especialmente importante para nuestra empresa, ya que tiene varias sedes en Alemania y España. La integración de este gemelo digital ha sido relativamente sencilla gracias al protocolo de comunicación diseñado, que permite el envío de los datos de la posición actual del robot de forma directa.

En la Ilustración 67 se puede observar al robot junto con una pantalla que reproduce el gemelo digital al fondo. En primer plano se ve un ordenador portátil ejecutando el HMI en la pantalla de control del robot.

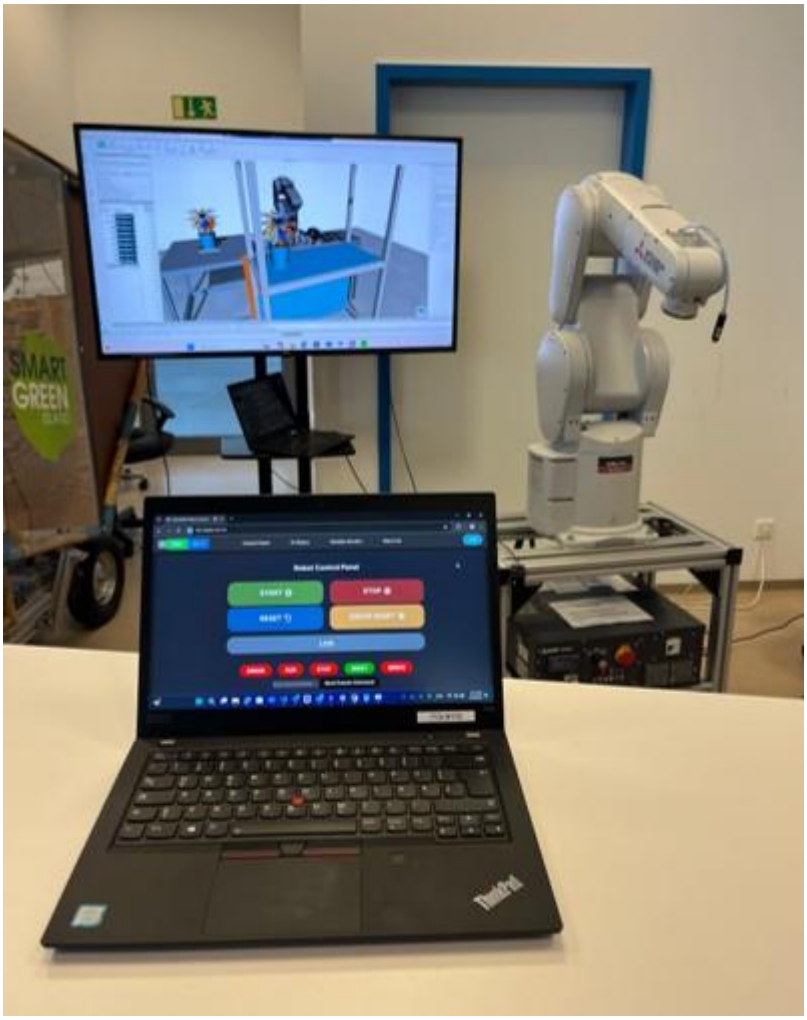

Ilustración 67: Robot Mitsubishi con el HMI ejecutado en un ordenador portátil y el Gemelo Digital reproducido en una pantalla.

5.1 Editor-Formateador en Tiempo Real MELFA-BasicV

5.1.1 Motivación

Mientras desarrollábamos el protocolo de comunicación y los programas del robot, nos encontramos con una gran dificultad al desarrollar programas en Melfa Basic V sin la licencia del software de pago proporcionado por Mitsubishi, ya que no disponíamos de ningún editor de código ni compilador y para poder probar los programas debíamos de primero programarlos en documentos de texto sin resaltado de código, luego introducirlos en un USB e introducirlos en el controlador del robot desde el USB. En caso de haber cometido un error al escribir el código solo podíamos saberlo al compilarlo en el controlador y teníamos que corregir el código y repetir el proceso.

Este método de trabajo resultaba tedioso para el desarrollo del software del robot, especialmente teniendo en cuenta que el código que el controlador

compila debe tener un formato muy específico, con las indentaciones, números de línea y símbolos oportunos.

Es por ello por lo que decidimos desarrollar nuestro propio editor web, no como un producto, si no como una herramienta para ayudarnos a desarrollar el software del robot.

5.1.2 Visión General

El **Editor-Formateador en Tiempo Real MELFA-BasicV** es una herramienta web personalizada diseñada para ayudar a los desarrolladores a escribir, formatear y resaltar código en el lenguaje de programación MELFA-BasicV (Ilustración 68). Este proyecto surgió de la necesidad de contar con una herramienta gratuita y accesible para desarrollar código MELFA sin necesidad de costosos paquetes de software oficiales. El editor permite a los usuarios escribir y formatear su código en tiempo real, proporcionando resaltado de sintaxis básico, características de formateo y la capacidad de descargar el código en formato .prg.

Este editor utiliza **Highlight.js** para el resaltado de sintaxis en tiempo real, una biblioteca de código abierto simple pero poderosa para resaltar sintaxis en la web.

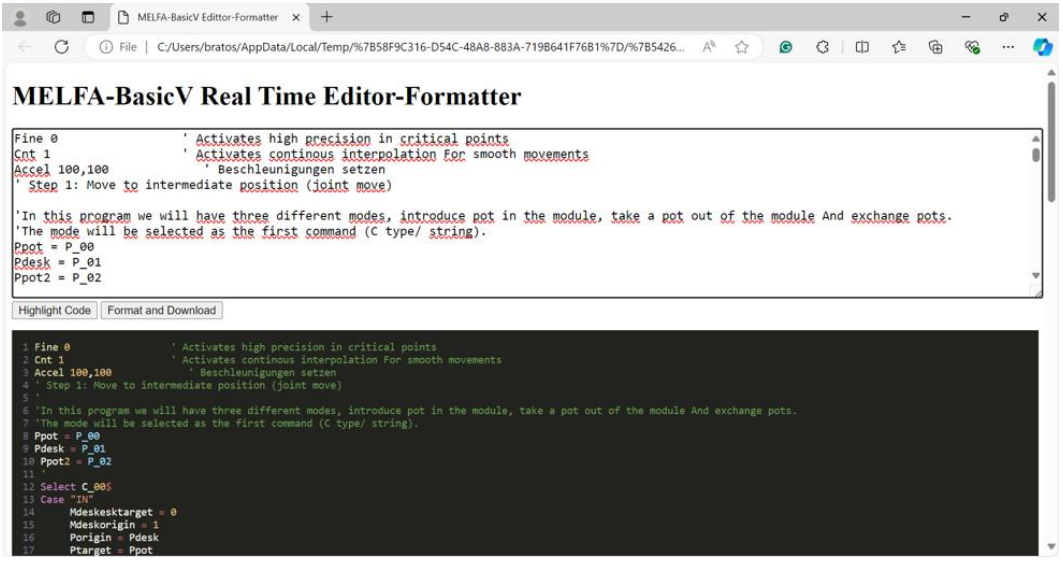

Ilustración 68: Editor-Formateador web en tiempo real para Melfa Basic V

5.1.3 Características

- **Entrada de Código en Tiempo Real**: Los usuarios pueden escribir código MELFA-BasicV directamente en el área de texto proporcionada.
- **Resaltado de Sintaxis**: Se proporciona retroalimentación visual instantánea al resaltar la sintaxis del código.

- **Formateo de Código**: El código se puede formatear automáticamente con números de línea, mejorando su legibilidad.
- **Exportación de Archivos**: El código formateado se puede descargar como un archivo .prg, adecuado para su uso en controladores de robots MELFA.
- **Atajos de Teclado**: Se dispone de un atajo (Ctrl + S) para guardar y resaltar el código rápidamente.
- **Estado Persistente**: La herramienta recuerda el último código escrito y la posición del cursor mediante el almacenamiento local del navegador, lo que permite a los usuarios continuar su trabajo incluso después de actualizar la página.

5.1.4 Características Adicionales

Además, se crearon dos scripts en Python, formatter.py e inverse_formatter.py. Estos scripts son útiles para convertir los programas formateados obtenidos del controlador en programas sin formato, permitiendo trabajar con ellos en nuestro editor.

5.1.5 Estructura del Código

Estructura HTML

El archivo HTML define la disposición de la página, incluyendo el área de texto para ingresar código, botones para resaltar y descargar el código, y un área de visualización para el código resaltado.

```
<!DOCTYPE html>
<html lang="es">
 <head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Editor-Formateador MELFA-BasicV</title>

  <!-- Incluir tema específico de Highlight.js para MELFA -->
  <link
   rel="stylesheet"
   href="https://zserub.github.io/MELFA-highlight.js/dist/melfa_dark.min.css
"
  />
 </head>
 <body>
  <h1>Editor-Formateador en Tiempo Real MELFA-BasicV</h1>
```

```
    <!-- Área de texto para ingresar código -->
    <textarea
      id="code-input"
      placeholder="Escribe código MELFA Basic aquí..."
      style="width: 100%; height: 200px; font-size: 16px;"
    ></textarea>

    <!-- Botones para activar resaltado y formateo -->
    <button onclick="highlightAndFormatCode()">Resaltar Código</button>
    <button onclick="formatAndDownload()">Formatear y Descargar</button>

    <!-- Área donde se mostrará el código resaltado -->
    <pre><code id="highlighted-code" class="language-melfa"></code></pre>

    <!-- Incluir highlight.js y el paquete de lenguaje MELFA -->
    <script src="https://cdnjs.cloudflare.com/ajax/libs/highlight.js/11.9.0/high
light.min.js"></script>
    <script src="https://zserub.github.io/MELFA-highlight.js/dist/melfa.min.js">
</script>
  </body>
</html>
```

- **Área de Texto**: Se utiliza <textarea> para ingresar el código MELFA-BasicV.
- **Botones**: Dos botones permiten al usuario activar el resaltado de código, formatear y descargar el código como un archivo .prg.
- **Integración con Highlight.js**: Se utiliza la biblioteca Highlight.js con un paquete de lenguaje personalizado para MELFA a fin de proporcionar resaltado de sintaxis.

Funcionalidad en JavaScript

El código JavaScript proporciona la funcionalidad principal, incluyendo el formateo de código, la gestión del almacenamiento local y la exportación de archivos.

1. **highlightAndFormatCode()**

Esta función guarda el código ingresado en el almacenamiento local y recarga la página para aplicar los cambios. Al recargar la página, se restaura el código guardado y la posición del cursor. Esto permite una experiencia fluida para el usuario con un resaltado en tiempo real.

```
function highlightAndFormatCode() {
  const textarea = document.getElementById("code-input");
```

```
  localStorage.setItem("code", textarea.value);
  const cursorPosition = textarea.selectionStart;
  localStorage.setItem("cursorPosition", cursorPosition);
  location.reload();
}
```

2. window.onload

Al cargar la página, esta función restaura el código previamente guardado y la posición del cursor desde el almacenamiento local. También aplica el resaltado de sintaxis utilizando la API de Highlight.js.

```
window.onload = function () {
  const savedCode = localStorage.getItem("code");
  const savedCursorPosition = localStorage.getItem("cursorPosition");

  if (savedCode) {
    const textarea = document.getElementById("code-input");
    const codeOutput = document.getElementById("highlighted-code");
    textarea.value = savedCode;

    if (savedCursorPosition !== null) {
      textarea.selectionStart = savedCursorPosition;
      textarea.selectionEnd = savedCursorPosition;
      textarea.focus();
    }

    const formattedCode = formatCode(savedCode);
    codeOutput.textContent = formattedCode;
    hljs.highlightElement(codeOutput);
  }
};
```

3. formatAndDownload()

Esta función formatea la entrada del usuario, solicita un nombre de archivo y descarga el código formateado como un archivo .prg.

```
function formatAndDownload() {
  const textarea = document.getElementById("code-input");
  const code = textarea.value;
  const formattedCode = formatCode(code);

  const fileName = prompt("Introduce el nombre del archivo (sin extensión):");

  if (fileName) {
    const blob = new Blob([formattedCode], { type: "text/plain" });
```

```
    const link = document.createElement("a");
    link.href = URL.createObjectURL(blob);
    link.download = `${fileName}.prg`;
    link.click();
  } else {
    alert("No se proporcionó un nombre de archivo.");
  }
}
```

4. formatCode()

Esta función formatea el código MELFA-BasicV agregando números de línea (necesarios para el compilador del controlador) y manejando correctamente los comentarios.

```
function formatCode(code) {
  const lines = code.split("\n");
  let insideComment = false;
  let result = "";
  let lineNumber = 1;

  lines.forEach((line) => {
    if (line.includes("/*")) {
      insideComment = true;
      line = line.replace("/*", "");
    }
    if (line.includes("*/")) {
      insideComment = false;
      line = line.replace("*/", "");
      result += line + "\n";
      return;
    }
    if (!insideComment) {
      if (line.trim() === "") {
        result += `${lineNumber} '\n`;
      } else {
        result += `${lineNumber} ${line}\n`;
      }
      lineNumber++;
    } else {
      result += line + "\n";
    }
  });
```

```
  return result;
}
```

Atajo de Teclado (Ctrl + S)

El código JavaScript también escucha la combinación de teclas Ctrl + S para activar la función highlightAndFormatCode(), lo que hace que la herramienta sea más rápida de usar para los desarrolladores que dependen de atajos de teclado.

```
document.addEventListener("keydown", function (event) {
  if (event.ctrlKey && event.key === "s") {
    event.preventDefault();
    highlightAndFormatCode();
  }
});
```

5.1.6 Resumen de Funcionamiento

1. **Entrada de Código**: El usuario escribe código MELFA-BasicV en el área de texto proporcionada.
2. **Resaltado de Código en Tiempo Real**: Cuando se realiza un cambio, el código se guarda, la página se actualiza y el código se resalta de manera fluida.
3. **Formateo y Descarga**: Al hacer clic en el botón “Formatear y Descargar”, el código se formatea, se le agregan números de línea y se exporta como un archivo .prg.
4. **Autoguardado**: El editor guarda automáticamente el código en el almacenamiento local del navegador y lo restaura al recargar la página.

5.1.7 Conclusión

El **Editor-Formateador en Tiempo Real MELFA-BasicV** es una alternativa de código abierto potente a las herramientas de desarrollo MELFA-BasicV de pago. Simplifica el proceso de programación para los desarrolladores al ofrecer resaltado de sintaxis en tiempo real, formateo de código y opciones convenientes de exportación de archivos, convirtiéndose en una herramienta valiosa para los desarrolladores de robótica MELFA.

5.2 Programas de Movimientos del Robot Mitsubishi Melfa RV-7FC-D

Este apartado describe los programas implementados en el robot **Mitsubishi Melfa RV-7FC-D** en **MELFA Basic V**.

Los programas analizados en este apartado se ejecutarán en el slot 1 del controlador, que es el único con permisos para realizar movimientos en el robot. Más adelante se explicarán más detalladamente las capacidades multitarea del controlador del robot, que permite tener hasta 8 'slots'.

Dichos programas permiten al robot realizar tres operaciones principales dentro del sistema mecatrónico automatizado de germinación de semillas:

1. **Extraer plantas del módulo de germinación**
2. **Introducir plantas en el módulo de germinación**
3. **Sacar fotos de las plantas**

Cada programa emplea una estructura modular con subrutinas reutilizables para optimizar el código y mejorar la eficiencia.

> Nota: el programa para la plantación de semillas todavía no está terminado, ya que primero debemos probar el correcto funcionamiento y tomar las medidas finales de la herramienta de sembrado, es por ello por lo que no se incluye en este apartado.

5.2.1 Programa TAKEPHOTO.prg: Tomar fotos de las plantas

Este programa mueve el robot a una posición específica para tomar una fotografía de la planta, permitiendo la supervisión y análisis de su estado.

Funcionamiento:

1. Se mueve el robot a una posición cómoda (**Pcomfy**).
2. Se desplaza hasta la posición de la maceta (**Ppot**).
3. Espera **3 segundos** para permitir la captura de la imagen.
4. Retorna a la posición cómoda.
5. Finaliza la ejecución.

Código:

```
Fine 0              ' Activa alta precisión en puntos críticos
Cnt 1               ' Activa interpolación continua para movimientos suaves
Accel 100,100       ' Ajusta aceleraciones
Spd 20
Ovrd 10

Pcomfy = P_39
Ppot = P_00

GoSub *MOVETOCOMFYPOS 'El robot se mueve hasta la posicion comoda
```

```
Mov Ppot, -150  ' El robot se mueve en direccion a la maceta con un offset de 150, para que la imagen muestre bien la maceta
Dly 3  ' Espera 3 segundos para tomar la foto

GoSub *MOVETOCOMFYPOS ' el robot vuelve a la posicion comoda
GoTo *THEEND

' --- Subrutinas ---
*MOVETOCOMFYPOS
Fine 50
Cnt 1
Ovrd 70
Mov Pcomfy
Return

*THEEND
Dly 1
M_20# = +1
End
```

5.2.2 Programa TAKEIN.prg: Introducir plantas en el módulo de germinación

Este programa permite transferir una planta desde la mesa de interacción con el humano hasta el módulo de germinación.

Funcionamiento:

1. Se define la posición de la maceta (**Ppot**) y de la mesa (**Pdesk**).
2. Se mueve el robot a la maceta, la recoge y vuelve a la posición de seguridad.
3. Se traslada la maceta al módulo de germinación y la deposita.
4. Se regresa a la posición cómoda.

Código:

```
Fine 0
Cnt 1
Accel 100,100

Ppot = P_00
Pdesk = P_01
Mdeskesktarget = 0
Mdeskorigin = 1
```

```
Porigin = Pdesk
Ptarget = Ppot

GoSub *PICKPOT
GoSub *LEAVEPOT
GoTo *THEEND

' --- Subrutinas ---
*PICKPOT
If Mdeskorigin = 1 Then
   Pcomfy = P_38
Else
   Pcomfy = P_39
EndIf
GoSub *MOVETOCOMFYPOS
Cnt 0
Fine 100
Spd 50
Mov Porigin,-100
Fine 60
Spd 30
Mvs Porigin
GoSub *MOVETOCOMFYPOS
Return

*LEAVEPOT
If Mdeskesktarget = 1 Then
   Pcomfy = P_38
Else
   Pcomfy = P_39
EndIf
GoSub *MOVETOCOMFYPOS
Cnt 0
Fine 100
Spd 50
Mov Ptarget,-100
Fine 60
Spd 30
Mvs Ptarget
GoSub *MOVETOCOMFYPOS
Return

*MOVETOCOMFYPOS
```

```
Fine 50
Cnt 1
Ovrd 80
Mov Pcomfy
Return

*THEEND
Dly 1
M_20# = +1
End
```

5.2.3 Programa TAKEOUT.prg: Extraer plantas del módulo de germinación

Este programa permite tomar una planta desde el módulo de germinación y trasladarla a la mesa.

Funcionamiento:

1. Se define la posición de la maceta (**Ppot**) y del escritorio (**Pdesk**).
2. Se toma la maceta del módulo de germinación.
3. Se traslada la maceta al escritorio y se suelta.
4. Se regresa a la posición cómoda.

Código:

```
Fine 0
Cnt 1
Accel 100,100

Ppot = P_00
Pdesk = P_01
Mdeskesktarget = 1
Mdeskorigin = 0
Porigin = Ppot
Ptarget = Pdesk

GoSub *PICKPOT
GoSub *LEAVEPOT
GoTo *THEEND

' --- Subrutinas ---
*PICKPOT
If Mdeskorigin = 1 Then
```

```
   Pcomfy = P_38
Else
   Pcomfy = P_39
EndIf
GoSub *MOVETOCOMFYPOS
Cnt 0
Fine 100
Spd 50
Mov Porigin,-100
Fine 60
Spd 30
Mvs Porigin
GoSub *MOVETOCOMFYPOS
Return

*LEAVEPOT
If Mdeskesktarget = 1 Then
   Pcomfy = P_38
Else
   Pcomfy = P_39
EndIf
GoSub *MOVETOCOMFYPOS
Cnt 0
Fine 100
Spd 50
Mov Ptarget,-100
Fine 60
Spd 30
Mvs Ptarget
GoSub *MOVETOCOMFYPOS
Return

*MOVETOCOMFYPOS
Fine 50
Cnt 1
Ovrd 80
Mov Pcomfy
Return

*THEEND
Dly 1
M_20# = +1
End
```

5.2.4 Conclusión

Los programas presentados utilizan subrutinas modulares para realizar tareas repetitivas de forma eficiente. Esto permite al robot ejecutar operaciones con mayor precisión, seguridad y eficiencia.

Características clave de la implementación:

- Uso de movimientos interpolados (Cnt) para transiciones suaves.
- Posiciones predefinidas obtenidas dinámicamente desde la Raspberry Pi (Pcomfy, Ppot, Pdesk).
- Subrutinas reutilizables (*MOVETOCOMFYPOS, *PICKPOT, *LEAVEPOT) para optimizar código.
- Manejo de señales (M_20# = +1) para comunicar que el programa ha terminado e informar a la Raspberry Pi.

5.3 Toma de Decisiones sobre la Comunicación y Presupuestos

Antes de comenzar el desarrollo del software, realizamos una investigación exhaustiva sobre todos los métodos posibles para comunicarnos con el robot y controlar las variables ambientales del módulo.

El factor principal que influyó en el desarrollo y dirección del proyecto fue el método de comunicación con el robot. Para ello, exploramos todas las opciones disponibles e identificamos cinco soluciones principales. Las primeras cuatro opciones dependían del control clásico mediante señales IO (requiriendo una tarjeta IO), mientras que la última opción utilizaba comunicación Ethernet, evitando el uso de señales IO del controlador.

5.3.1 Opciones 1 y 2: Construcción de Nuestro Propio Módulo Electrónico

La primera opción consistía en construir un módulo electrónico personalizado. En este enfoque, se adquirirían componentes electrónicos individuales y se crearían todas las conexiones dentro de un gabinete eléctrico grande.

- **Opción 1**: Utilizaba un control de 16 IO.
- **Opción 2**: Utilizaba un control de 32 IO.

Este método permitiría evitar el uso de un PLC.

Ventajas:

- Más económico que otras soluciones basadas en tarjetas IO.

- No requiere el uso de un PLC.

Desventajas:

- Implementación significativamente más lenta.
- Menor fiabilidad en comparación con otras soluciones basadas en tarjetas IO.
- Funcionalidad limitada debido a la dependencia de señales IO.
- Requiere cableado y conexiones extensas.

5.3.2 Opción 3: Compra de un Módulo Industrial Raspberry Pi

Esta opción consistía en adquirir un módulo industrial basado en Raspberry Pi, fabricado por un proveedor, proporcionando una solución altamente fiable sin necesidad de un PLC.

Ventajas:

- Alta fiabilidad.
- No requiere el uso de un PLC.

Desventajas:

- Solución costosa.
- Funcionalidad limitada debido a la dependencia de señales IO.
- Requiere cableado y conexiones extensas.

5.3.3 Opción 4: Uso del PLC Más Económico Disponible

En esta opción se adoptaba una solución tradicional basada en PLC, utilizando una interfaz de señales IO entre el PLC y el controlador del robot. Se optaría por **el PLC más barato del mercado**, aunque también se consideró el uso de un PLC de repuesto si estuviera disponible.

Ventajas:

- Alta fiabilidad.
- Solución comúnmente utilizada en entornos de producción.
- Control en tiempo real.

Desventajas:

- Solución costosa.
- Funcionalidad limitada debido a la dependencia de señales IO.

- Requiere cableado y conexiones extensas.

5.3.4 Opción 5: Comunicación Ethernet (IoT) con Raspberry Pi

Esta opción utiliza comunicación Ethernet directamente entre la Raspberry Pi y el controlador del robot, eliminando la necesidad de un PLC o señales IO para controlar el robot.

Ventajas:

- Solución más económica (1/10 del coste de las otras opciones).
- No requiere el uso de un PLC.
- Alta escalabilidad (la escalabilidad depende del software en lugar del hardware).
- Control con IA: Permite coordinación basada en inteligencia artificial.
- Gemelo Digital: Permite simulaciones con datos reales del robot.
- Mínimo cableado (solo un cable de comunicación).
- Alta versatilidad, adaptable a diversas tareas y entornos.
- Tiempo de comunicación rápido (15ms).
- Acceso remoto: Posibilidad de controlar el robot a través de la red local.

Desventajas:

- No es una solución en tiempo real, lo que podría ser una limitación en aplicaciones industriales.

5.3.5 Presupuesto para Soluciones de Comunicación

En la Ilustración 69 se puede ver el presupuesto realizado para las distintas opciones de comunicación con el robot. Esta imagen se encuentra ampliada en el Anejo 4.

	BUDGET FOR COMMUNICATION SOLUTION															
		Option 1	Option 2	Option 3	Option 4	Option 5		Option 1	Option 2	Option 3	Option 4	Option 5	Supplier	LINKS
	Item	Nº 8 I/8 O	Nº 16 I/16 O	Nº Revpi	Nº C_PLC	Nº Eth-TCP/IP	Price	Cost 8 I/8 O	Cost 16 I/16 O	Cost Revpi	Nº C_PLC	Nº Eth-TCP/IP		
Cards	2D-TZ378		1	1	1	1	€ 442.00	€ 442.00	€ 442.00	€ 442.00	€ 442.00	€ -	ROBTEC	https://www.robte
	2D-CBL05 Connection Cable		1	1	1	1	€ 324.06	€ 324.06	€ 324.06	€ 324.06	€ 324.06	€ -		
	RASP PI 4 B 8GB		1	1		1	1 € 79.80	€ 79.80	€ 79.80	€ -	€ 79.80	€ 79.80	Reichelt	https://www.reich
Raspy Modules	TCA9548A 8 I2C multiplexor		1	1			€ 2.80	€ 2.80	€ 2.80	€ -	€ -	€ -	Reichelt	https://www.reich
	8-relay 3,3V up to 230V		1	2			1 € 12.00	€ 12.00	€ 24.00	€ -	€ -	€ 12.00	Amazon	https://www.amaz
	Optocoupler 8-Channel 24V to 3,3V		1	2			1 € 17.00	€ 17.00	€ 34.00	€ -	€ -	€ 17.00	Amazon	https://www.amaz
	MCP23017- IO 16 Digital		1	2			€ 6.90	€ 6.90	€ 13.80	€ -	€ -	€ -	Botland	https://botland.de
Power supply	120 W 24 V 5 A Power Supply		1	1	1	1	€ 29.50	€ 29.50	€ 29.50	€ 29.50	€ 29.50	€ -	Amazon	https://www.amaz
	Raspberry Pi - power supply, 5.1 V, 5.0 A,		1	1		1	1 € 12.65	€ 12.65	€ 12.65	€ -	€ 12.65	€ 12.65	Reichelt	https://www.reich
RevPi	KUNBUS Revolution PI Control Core S 16 GB PR100360				1		€ 291.89			€ 291.89	€ -	€ -	Pollin	https://www.pollin
	Digital I/O module 14 I / 14 O				1		€ 218.30			€ 218.30	€ -	€ -	Reichelt	https://www.reich
CLICK PLC	PLC					1	€ 127.00				€ 127.00	€ -	PLC Direct Benelus	https://www.picdi
	DC Inputs					1	€ 66.00				€ 66.00	€ -	PLC Direct Benelus	https://www.picdi
	DC Outputs					1	€ 72.00				€ 72.00	€ -	PLC Direct Benelus	https://www.picdi
Cable	D2-DSCBL					1	€ 21.60				€ 21.60	€ -	PLC Direct Benelus	https://www.picdi
	USB-A converter, 1x RS232					1	€ 11.58				€ 11.58	€ -	Reichelt	https://www.reich
	Total wihout 2D-TZ378 (mandatory)							€ 160.65	€ 196.55	€ 539.69	€ 420.13	€ 121.45		
	Total							€ 926.71	€ 962.61	€ 1,305.75	€ 1,186.19	€ 121.45		
	Time cost							Cost 1,5 Weeks 2P work		Instant	Instant	1 week 2P work		

Ilustración 69: Presupuesto para las distintas opciones de comunicación con el robot.

5.3.6 Comparación de Presupuesto

En la Tabla 9 se puede ver la comparación de los presupuestos de cada una de las posibles soluciones para realizar la comunicación con el robot.

Tabla 9: Comparación de los presupuestos para las diferentes opciones para la comunicación con el robot.

Solution	Budget
Make our Electronic Module	962,61 EUR
Buy a Raspberry Pi Industrial Module	1305,75 EUR
Use the cheapest PLC available.	1186,19 EUR
Use Ethernet(IoT)-RaspberryPi.	121,45 EUR

5.4 Comunicación entre el Controlador del Robot y la Raspberry Pi

5.4.1 Introducción

Este apartado explica el sistema de comunicación entre un servidor, desarrollado en **Python** utilizando **FastAPI**, y el controlador del robot Mitsubishi programado en **Melfa Basic V**. El objetivo del sistema es permitir que el servidor gestione y controle el robot enviando comandos a través de un socket TCP, mientras que el controlador responde ejecutando comandos específicos que influyen en el comportamiento del robot.

El sistema está diseñado para operar de manera asíncrona y proporciona una interfaz accesible desde una aplicación web u otros clientes a través de WebSockets. El servidor actúa como intermediario, enviando comandos al controlador, procesando respuestas y transmitiendo el estado actual del robot mediante una conexión persistente.

Este método de comunicación es en gran medida gracias a las capacidades multitarea del controlador del robot. El controlador permite la ejecución de hasta 8 slots diferentes simultáneamente, de los cuales se ejecuta una cantidad configurable de líneas de código de forma secuencial, permitiéndose así fijar prioridades.

Para evitar que varios slots traten de ejecutar movimientos en el robot a la vez, el fabricante ha limitado los permisos de movimiento del robot al slot 1, pero permite totalmente que otros slots carguen, descarguen, ejecuten, paren o

reseteen programas en los otros slots. Cualidad muy importante para el desarrollo de nuestra lógica de control y protocolo de comunicación.

5.4.2 Explicación Detallada sobre la Comunicación Ethernet del Controlador

Esta sección describe cómo establecer comunicación entre un robot Mitsubishi (serie de controladores CR750/CR751) y un servidor externo, como una Raspberry Pi, utilizando el método de enlace de datos Ethernet. El proceso incluye la configuración de la conexión, el código básico en MELFA BASIC V para enviar y recibir datos, y la comprensión del protocolo de comunicación tanto para el robot como para la programación del lado del servidor.

1. Resumen de la Configuración de Comunicación

Los controladores Mitsubishi CR750/CR751 soportan comunicación Ethernet TCP/IP, lo que permite un intercambio de datos eficiente con dispositivos externos, como PCs, PLCs o servidores. Esto permite controlar remotamente el robot, monitorear su estado o intercambiar datos entre el robot y sistemas externos.

Para establecer un enlace de datos Ethernet, es necesario configurar tanto el controlador del robot como el servidor externo (por ejemplo, una Raspberry Pi). Los componentes clave son:

- **Controlador CR750/CR751**: La unidad de control del robot, responsable de gestionar los comandos y ejecutar programas.
- **Servidor Externo**: Un dispositivo que envía/recibe datos a través de la red hacia/desde el robot (por ejemplo, Raspberry Pi).
- **Red TCP/IP**: Es necesaria una conexión Ethernet entre el controlador del robot y el servidor externo.

2. Configuración del Controlador del Robot para Comunicación Ethernet

Paso 1: Conexión de Hardware

Asegúrese de que el controlador CR750/CR751 esté conectado a la misma red local que su servidor externo a través de Ethernet. El controlador del robot generalmente tiene un puerto de red que se puede utilizar para este propósito. Verifique su enrutador o switch de red para confirmar que ambos dispositivos están en la misma subred.

Paso 2: Configuración de la Dirección IP del Robot

5. **Acceda a la Configuración de Red del Controlador del Robot:**
    - Configure los ajustes de red del controlador CR750/CR751 mediante el dispositivo de enseñanza (*teaching pendant*).

- Asigne una dirección IP estática al controlador del robot. Esta dirección debe ser única en su red para evitar conflictos.
- Por ejemplo, asigne la dirección IP 192.168.0.20 al robot.
- Configure las siguientes propiedades para COM2:
  - Netport: 10002
  - CPRCE12: 2
  - COMDEV: OPT12
- Configure las siguientes propiedades para COM3:
  - Netport: 10003
  - CPRCE12: 3
  - COMDEV: OPT13

6. **Configure la Dirección IP del Servidor Externo:**
   - Asegúrese de que el servidor externo (por ejemplo, una Raspberry Pi) también tenga una dirección IP estática dentro de la misma subred, como 192.168.0.10.
7. **Confirme la Conectividad:**
   - Después de configurar las direcciones IP, pruebe la conectividad de la red utilizando el comando ping desde el servidor (por ejemplo, Raspberry Pi):

     ping 192.168.0.20
   - Asegúrese de recibir respuestas del controlador del robot, lo que indica que la conexión está activa.

3. Programación de Comunicación Ethernet con MELFA BASIC V

Para habilitar la comunicación del robot a través del enlace Ethernet, es necesario escribir un programa en **MELFA BASIC V**. A continuación, se explica en detalle cómo funciona el código proporcionado en MELFA BASIC V.

Paso 1: Abrir la Línea de Comunicación Ethernet

El siguiente comando establece un canal de comunicación a través de Ethernet:

```
Open "COM2:" As #1
```

- COM2: Especifica el puerto Ethernet.
- As #1: Se refiere al canal de entrada/salida abierta para la comunicación.

Paso 2: Enviar y Recibir Datos

Una vez abierta la línea de comunicación, se pueden enviar y recibir datos utilizando los comandos Print e Input.

- **Enviar Datos** (como una respuesta) al Cliente:

  Print #1, "SELECT";" "

  Esta línea envía la cadena "SELECT" al servidor externo como respuesta a una solicitud previa.

- **Recibir Datos del Cliente**:

  Input #1, M1

  Este comando escucha la entrada desde el servidor (o dispositivo externo) y almacena el valor recibido en la variable M1.

Paso 3: Procesar el Comando

En función del valor recibido (M1), el programa toma la acción correspondiente. Esto se gestiona mediante la declaración Select:

Select M1

- **Opciones de Comando**: El programa soporta múltiples comandos, cada uno representado por un caso. La lógica de la comunicación y el procesamiento de comandos se explicará en otra sección.

Paso 4: Cerrar la Conexión

Cuando el programa finaliza, la línea de comunicación se cierra para garantizar un manejo adecuado de la conexión:

Close #1

4. Programación de la Interfaz de Robot en Python

La interfaz del robot en Python es un módulo que permite la comunicación con el robot utilizando una conexión de socket TCP/IP y un método de “handshake”. A continuación, se detalla el funcionamiento del código proporcionado para esta interfaz.

Paso 1: Crear una Conexión de Socket TCP

La interfaz debe abrir un socket para comunicarse con el controlador del robot. El código en Python para crear un socket se ve así:

```
self.robot_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
await asyncio.get_running_loop().sock_connect(self.robot_socket, (CONTROLL
ER_IP, CONTROLLER_PORT))
```

Aquí, CONTROLLER_IP se refiere a la dirección IP del robot, y CONTROLLER_PORT es el número de puerto configurado para la comunicación (por ejemplo, 10002).

Paso 2: Enviar Comandos al Robot

La interfaz envía diferentes tipos de datos (cadenas, enteros, etc.) a través del socket TCP. Por ejemplo, para enviar un número y entrar en el modo de selección de programa en el programa de comunicación del servidor del robot:

```
self.robot_socket.sendall("1".encode('utf-8'))
```

- La cadena "1" se envía al robot, indicando que el menú de comunicación del robot (que siempre se ejecuta en un bucle) debe entrar en “Modo de Selección de Programa”.

Paso 3: Recibir Respuestas del Robot

La interfaz de Python siempre escucha las respuestas del controlador del robot en un bucle secundario:

```
self.receiver_thread = threading.Thread(target=receive_messages, args=(self.
robot_socket, self.state_update_event))
self.receiver_thread.start()
```

Aquí se crea y se inicia un hilo que siempre escucha los mensajes del robot.

```
data = sock.recv(1024)
decoded_data = data.decode('utf-8')
```

La función recv() lee la respuesta del robot, y decode('utf-8') la convierte en un formato legible. Algunos de estos mensajes se almacenarán en una cola. Más adelante se explicará cómo se gestionan estos mensajes.

5. Manejo y Validación de Errores

Manejo de Errores de Conexión (Python)

En el lado de la interfaz Python, los errores como tiempos de espera o fallos en la comunicación se manejan utilizando bloques try-except:

```
try:
    self.robot_socket.sendall(command.encode('utf-8'))
except Exception as e:
    print(f"Error: {e}")
```

En caso de error, el servidor intenta recuperar la conexión o alerta al usuario.

5.4.3 Comunicación Entre la Aplicación Python y el Controlador (Lado del Controlador)

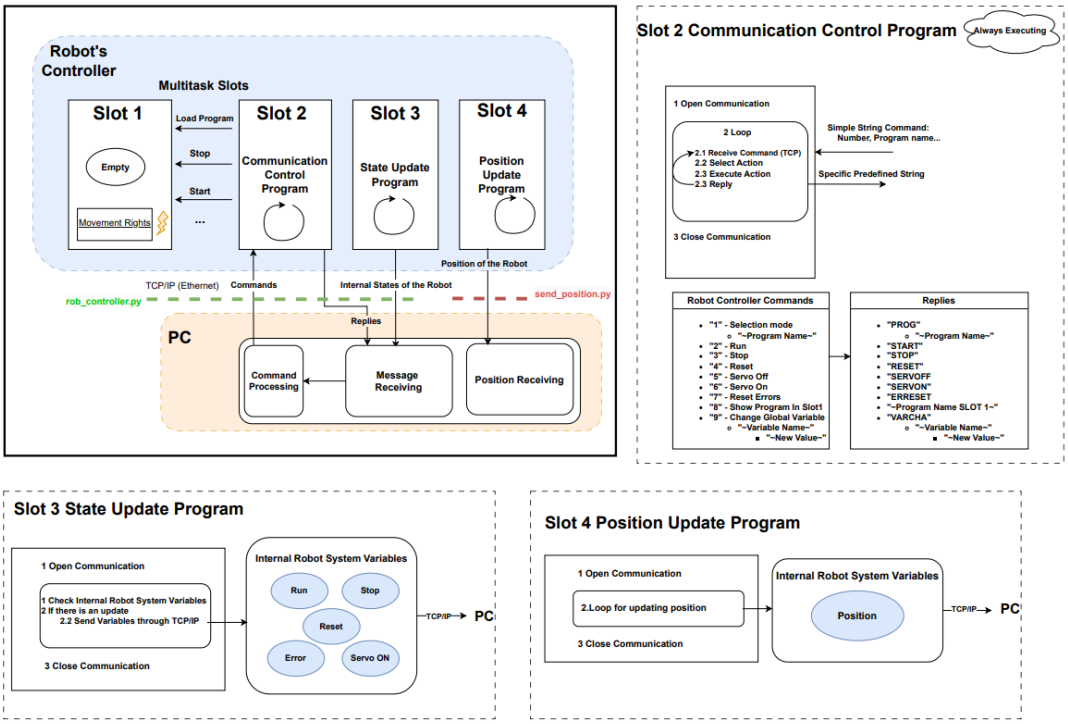

Ilustración 70: Diagrama de la arquitectura del software multitarea del robot y su comunicación con la Raspberry Pi(PC)

En la Ilustración 70 (que se encuentra ampliada en el Anejo 4) se puede ver la arquitectura del software multitarea del controlador del robot y la comunicación con la Raspberry Pi, que se comporta como un PC.

Componentes Principales

El sistema consta de dos partes principales:

1. **Controlador del Robot Mitsubishi (Melfa Basic V):**

   Programas escritos en **Melfa Basic V** que se ejecuta directamente en los diferentes slots del controlador del robot.

   - Slot 2: Gestiona la recepción de comandos a través de un socket de comunicación Ethernet y ejecuta los comandos correspondientes a los programas y funciones del robot.
   - Slot 3: obtiene las variables de estado del robot y se las envía al backend a través del mismo socket de comunicación a través del cual se envían los comandos.
   - Slot 4: obtiene la posición real del robot y se la envía al servidor a través de un socket diferente que se utiliza únicamente para este propósito.

2. **Interfaces del Robot en Python**:

- rob_controller.py:
    - Se comunica con el controlador mediante un socket TCP.
    - Transforma comandos de alto nivel en secuencias de mensajes TCP.
    - Gestiona la recepción de estados del robot.
    - Admite la lectura de registros, el envío de comandos y la consulta de estados.
    - Integra una API utilizando **FastAPI** para la gestión de una aplicación web.
- send_position:
    - Se comunica con el controlador mediante un socket TCP.
    - Obtiene la posición del robot a través del socket destinado a este fin.
    - Envía dicha posición al software del Gemelo Digital (ISG-Virtuos) a través de un socket diferente, destinado a la comunicación con el Gemelo Digital.

Protocolo de Comunicación

Hemos desarrollado un protocolo de comunicación sencillo basado en un sistema de *handshake*. El aspecto principal de este sistema de comunicación es que cuando el cliente envía un comando al robot a través de TCP, espera recibir una respuesta del robot. Esta respuesta debe ser concreta y estar predefinida, de manera que corresponda de manera inequívoca al comando enviado por el cliente. Este sistema permite controlar el estado y el flujo del sistema de manera sencilla y es fácilmente escalable, como se detallará en la documentación posterior.

Lógica de los Comandos

Se han desarrollado un conjunto de funciones base, que son las interacciones de más bajo nivel con el controlador. También se han implementado *proto-comandos*, que son interacciones de nivel intermedio que se ejecutan en la aplicación Python y que, a su vez, utilizan funciones base. Adicionalmente, se diseñaron comandos complejos que ejecutan *proto-comandos*, de manera similar a como los *proto-comandos* ejecutan funciones base.

Dado que ahora estamos analizando la comunicación en el lado del controlador, es importante destacar que el controlador únicamente recibe y

envía información en formato de cadenas de texto (*strings*), que posteriormente pueden almacenarse como enteros o datos de posición. En esta sección de la documentación se detalla la lógica que sigue el controlador una vez que recibe estas cadenas de texto.

Gestión de Comandos en el Controlador

La gestión de comandos en el controlador se lleva a cabo en el slot 2, que ejecuta de forma continua un programa llamado menu.prg, el cual analizaremos a continuación:

1. Recepción de Comandos en el Controlador:

- El controlador espera comandos del servidor en un bucle continuo que se ejecuta siempre en el slot 2 del controlador:

```
*R
Print #1, "SELECT";" " ' Transmite la cadena de caracteres "SELECT" par a solicitar una entrada
Input #1, M1  ' Recibe la entrada del servidor y la guarda en M1
'
' Seleccionamos una opción dependiendo del valor de M1
Select M1
... (Aquí va todo el código del árbol de decisiones dentro del menú)...
GoTo *R  ' Bucle que sigue recibiendo comandos
```

- Dependiendo del comando recibido (almacenado en la variable M1), el controlador ejecuta acciones específicas como iniciar, detener programas o cambiar variables.

2. Comandos en el Controlador:

Aquí se enumeran todos los comandos que el controlador puede entender:

- **Seleccionar un Programa para Ejecutar en el Slot 1**:

```
Case 1
Print #1, "PROG ",  ' Enviar "PROG" para indicar que se seleccionará un programa
Input #1, C1$  ' Recibir entrada para la selección del programa
Select C1$  ' Seleccionar el programa basado en la entrada
Case "PRCHAVAR"
  XLoad 1, "PRCHAVAR"  ' Cargar el programa "PRCHAVAR" en el slot 1
  Print #1, "PRCHAVAR ",  ' Imprimir el programa seleccionado
```

- **Inicio del Programa**:

  Si se recibe el valor 2 en M1, el controlador inicia un programa específico cargado en el robot:

```
Case 2
Print #1, "START ",  ' Imprimir comando de inicio
If M_Run(1) = 0 Then ' Comprobar si el programa en el slot 1 no está e
n ejecución
  XRun 1  ' Ejecutar el programa en el slot 1
EndIf
```

- **Detención del Programa:**

  Si se recibe el comando STOP, el controlador detiene el programa en ejecución:

```
Case 3
Print #1, "STOP "; " "  ' Imprimir comando de parada
XStp 1  ' Detener el programa
```

- **Reinicio del Programa:**

  Si se recibe el comando RESET, el controlador reinicia el programa actual en el slot 1:

```
Case 4
Print #1, "RESET "; " "  ' Imprimir comando de reinicio
If M_Run(1) = 1 Then ' Si el programa en el slot 1 está en ejecución
  XStp 1  ' Detener el programa
  Wait M_Wai(1) = 1  ' Esperar hasta que el programa se detenga
EndIf
XRst 1  ' Reiniciar el programa en el slot 1
```

- **Apagar Servos:**

  Si se recibe el comando SERVOFF, el controlador carga un programa en el slot 1 que apaga los servos:

```
Case 5
Print #1, "SERVOFF ",  ' Imprimir comando de apagado de servos
XLoad 1, "SERVON"  ' Cargar el programa "SERVON"
Dly 1  ' Esperar 1 segundo
XRun 1  ' Ejecutar el programa "SERVON"
Dly 2
```

- **Encender Servos:**

  Si se recibe el comando SERVON, el controlador carga un programa en el slot 1 que enciende los servos:

```
Case 6
Print #1, "SERVON ",  ' Imprimir comando de encendido de servos
```

```
XLoad 1, "SERVON"  ' Cargar el programa "SERVON"
Dly 1  ' Esperar 1 segundo
XRun 1  ' Ejecutar el programa "SERVON"
Dly 2
```

- **Restablecer Errores**:

  Si se recibe el comando ERRESET, el controlador verifica si hay un error activo y lo reinicia:

```
Print #1, "ERRESET ",  ' Imprimir comando de restablecimiento de error
es
If M_Err = 1 Then Reset Err  ' Restablecer el error si está presente
```

- **Imprimir Programa Actual**:

  El controlador imprime el nombre del programa que está actualmente cargado en el slot 1:

```
Case 8
Print #1, C_Prg(1); " "  ' Imprimir el programa actual en el slot
```

- **Modificar una Variable Local del Controlador**:

  Se llama a una subrutina responsable de modificar variables globales del controlador:

```
Case 9
Print #1, "VARCHA ",  ' Imprimir comando de cambio de variable
GoSub *CHANGEVARIABLE
```

  La subrutina solicita el nombre de la variable a modificar y luego su valor:

```
*CHANGEVARIABLE
Input #1, C2$  ' Recibir entrada para la selección de la variable
Select C2$  ' Seleccionar la variable a cambiar según la entrada
' Variables de posición P_00 a P_04
Case "P_00"
  Print #1, "P_00",
  Input #1, P00  ' Variable local temporal
  P_00 = P00  ' Cambiar el valor de P_00
  Print #1, "CHANGED ",
Case "P_01"
  Print #1, "P_01",
  Input #1, P00
  P_01 = P00
```

```
  Print #1, "CHANGED ",
...
```

Gestión de Estados

1. Actualización de Estados en el Controlador:

- En el slot 3, donde se ejecuta el programa STATECHB.prg, el controlador monitorea continuamente cambios en variables clave como M_Err, M_Run(1), M_Wai(1), M_Psa(1), y M_Svo(1) para rastrear el estado del robot.

- El programa se ejecuta en un bucle en un slot distinto al slot 1 (y al del menú), comparando los valores actuales de estas variables con sus estados anteriores. Si se detecta un cambio, se activa una bandera y, al final del bucle, si la bandera es positiva, se envía el estado actualizado al servidor:

  ```
  Print #1, "STATE", M_Err, M_Run(1), M_Wai(1), M_Psa(1), M_Svo(1)
  ```

  Esto permite que el servidor reciba y procese actualizaciones del estado en tiempo real de manera asíncrona.

2. Actualización de Estados en la Interfaz Python:

- El controlador actualiza el estado del robot (ejecución del programa, errores, etc.). La interfaz Python recibe estos datos en el hilo encargado de recibir datos del controlador:

  ```
  await robot_controller.state_update_event.wait()
  response = await robot_controller.get_state()
  ```

Gestión de Posición

1. Actualización de Posición en el Controlador:

En el slot 4, ejecutando el programa POSITION.prg, el controlador transmite constantemente la posición del robot, lo que permite obtener la posición real del robot para la simulación.

El programa se ejecuta en un bucle en un slot diferente al slot 1, o a los slots donde se ejecutan el Menú o el programa de Actualización de Estado. Este programa abre un nuevo canal de comunicación y envía los datos de posición al servidor:

```
Open "COM3:" As #2 'Abrir línea Ethernet
*R
Wait M_Run(1)=1
Print #2, J_Curr
Dly(0.01)
```

```
GoTo *R
End
```

Esto permite que el servidor reciba y procese la posición real del robot.

2. Actualización de Posición en la Interfaz de Python del Robot y Envío al Gemelo Digital:

El controlador actualiza la posición del robot. La Interfaz en Python recibe estos datos en otro hilo de Python a través de un nuevo socket. Luego, actualiza la posición y la reenvía al Software de Gemelo Digital (ISG) a través del socket establecido entre el servidor y el software ISG.

5.4.4 Gestión de Comunicación y Comandos en la Interfaz Python con el Robot Mitsubishi Melfa

El backend de la aplicación web, construido utilizando FastAPI, implementa diversas funcionalidades para comunicarse con el robot y actualizar los estados mediante WebSockets. Los métodos clave gestionan los comandos y el estado del robot.

Esta documentación explica cómo la biblioteca en Python rob_controller.py gestiona la comunicación entre un servidor FastAPI y el robot Mitsubishi Melfa. Se centra en el proceso de enviar y recibir comandos, manejar comandos complejos con argumentos definidos por el usuario y utilizar bibliotecas JSON externas para definir comandos.

1. Capa de Comunicación: Lógica Detrás del Envío y Recepción de Comandos

La comunicación entre el servidor FastAPI y el robot Mitsubishi Melfa se gestiona a través de la biblioteca rob_controller.py.

La biblioteca **rob_controller.py** maneja probablemente la comunicación de bajo nivel y la traducción de comandos desde solicitudes HTTP hacia instrucciones legibles para el robot. Esto incluye comunicación por sockets o protocolos de comunicación serial típicamente utilizados con robots industriales.

Para la gestión de esta comunicación, hemos desarrollado las siguientes características:

- **Funciones base**: como receive y send, en las cuales se basa toda la comunicación. Estas funciones se utilizan para enviar y recibir mensajes usando sockets y comunicación TCP.
- **Función send_and_receive**: está compuesta por las funciones mencionadas anteriormente y es la función base para el protocolo de comunicación de *handshake*. Envía datos desde la aplicación Python y hace que espere hasta recibir una respuesta.

- **Proto-comandos**: un ejemplo de proto-comando sería program_select(sock, program_name). Esta función está compuesta por dos llamadas a send_and_receive. En la primera, se envía el comando "1" para indicar al controlador que queremos entrar en el modo de selección de programa. Luego, se espera el *handshake*, que sería la cadena "PROG". Después de esto, se envía el nombre del programa (program_name) y se espera la respuesta del *handshake* una vez que el programa ha sido cargado. En este caso, la respuesta sería el mismo nombre del programa.

- **Comandos complejos**: comandos que realizan varias llamadas a proto-comandos mientras controlan cuidadosamente el estado y el flujo del sistema. Todas estas funciones de proto-comandos devuelven True o False, de modo que la interfaz Python puede confirmar la ejecución correcta. De esta manera, el siguiente proto-comando en una secuencia solo se ejecutará si el anterior se ha ejecutado con éxito. Estos comandos complejos nos permiten programar a un nivel mucho más alto, habilitando el desarrollo de software complejo de manera más rápida y eficiente, controlando errores y asegurando una ejecución y comunicación correctas.

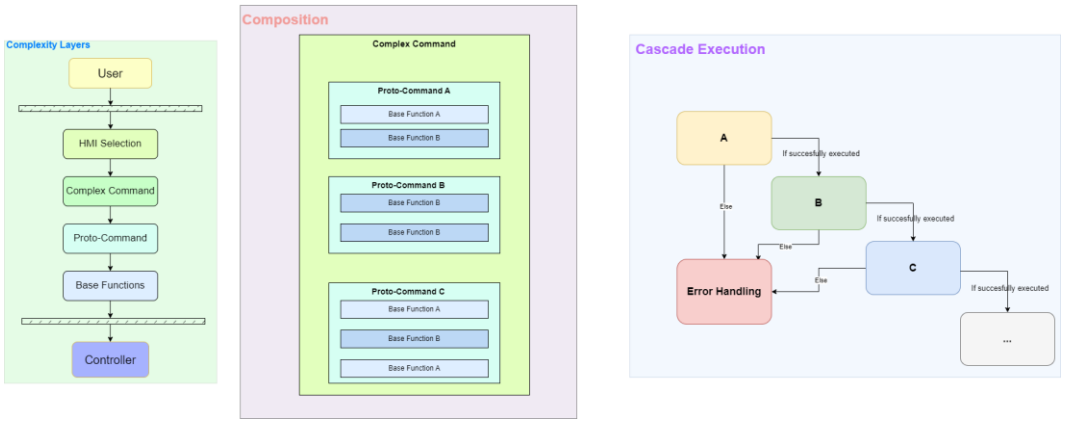

Ilustración 71: Diagrama que muestra las capas de complejidad en orden descendente, un ejemplo de composición de comandos y la ejecución en cascada de los comandos.

Todos los comandos del mismo tipo, independientemente de si son funciones base, proto-comandos o comandos complejos, se ejecutan en forma de cascada, y solo si el anterior comando del mismo tipo se ejecutó de una forma satisfactoria. En caso de error se para la ejecución y se gestiona dicho error.

En la Ilustración 71 se puede observar un diagrama que muestra las distintas capas de complexidad, la composición encapsulada de los comandos y la ejecución en cascada.

2. Gestión de Comandos Complejos con Argumentos Definidos por el Usuario

Algunos comandos enviados al robot requieren argumentos específicos proporcionados por el usuario para ejecutar acciones más complejas. Estos argumentos pueden ser posiciones, movimientos u operaciones predefinidas codificadas en los archivos JSON adjuntos. Desglosemos cómo se estructuran y procesan estos comandos complejos desde el nivel más bajo al más alto.

2.1 Funciones Base

Las funciones base representan las acciones de comunicación de bajo nivel. Desglosaremos cómo funcionan las funciones base y cómo se utilizan.

2.1.1 send_command(sock, command)

Propósito:

Envía una cadena de comando al controlador del robot mediante un socket TCP.

Detalles de Implementación:

- **Codificación y Transmisión del Comando:**

  sock.sendall(command.encode('utf-8'))

  - La cadena command se codifica en bytes utilizando codificación UTF-8.
  - sock.sendall() envía todos los bytes codificados a través del socket TCP sock. Este método garantiza que todos los datos se envíen antes de devolver el control.

- **Manejo de Excepciones:**

```
except Exception as e:
    print(f"Error: Could not send the command: {e}")
    if not enter_error_mode(sock, "TIMEOUT"):
        raise ExecuteError(f" Communication error: Lost Communication (S
END)")
```

  - Captura cualquier excepción que pueda ocurrir durante el proceso de envío.
  - Registra un mensaje de error con los detalles de la excepción.
  - Llama a enter_error_mode(sock, "TIMEOUT") para manejar la situación de error (se asume que la implementación de enter_error_mode está definida en otro lugar).
  - Si enter_error_mode devuelve False, lanza un ExecuteError, indicando una falla crítica en la comunicación.

2.1.2 recv_response(sock, expected_response)

Propósito:

Recibe una respuesta del controlador del robot y verifica si coincide con la respuesta esperada.

Detalles de Implementación:

- **Recepción de la Respuesta:**

  ```
  response = msg_queue.get(timeout=5)
  ```

  - Intenta recuperar un mensaje de respuesta de una cola de mensajes msg_queue.
  - El parámetro timeout=5 especifica que la función esperará hasta 5 segundos para una respuesta antes de lanzar una excepción queue.Empty.
  - **Nota:** Se asume que msg_queue es una cola segura para hilos (por ejemplo, queue.Queue) que es llenada por otro proceso o hilo que lee datos entrantes del socket.

- **Validación de la Respuesta:**

  ```
  if response == expected_response:
      return True
  else:
      raise ExecuteError(f" Incorrect response -> Expected:{expected_resp
  onse} vs Received:{response}  ")
  ```

  - Compara la respuesta recibida response con la respuesta esperada expected_response.
  - Si coinciden, devuelve True, indicando una comunicación exitosa.
  - Si no coinciden, lanza un ExecuteError con detalles sobre la discrepancia.

- **Manejo de Excepciones:**

  ```
  except queue.Empty:
      if not enter_error_mode(sock, "TIMEOUT"):
          raise ExecuteError(f" Communication error: Lost Communication (
  RECEIVE)")
      return False
  ```

  - Captura la excepción queue.Empty, que indica que no se recibió una respuesta dentro del período de espera.

- Llama a enter_error_mode(sock, "TIMEOUT") para manejar la situación de tiempo de espera.
- Si enter_error_mode devuelve False, lanza un ExecuteError.
- Devuelve False para indicar un error al recibir la respuesta.

2.1.3 send_and_receive(sock, command, expected_response)

Propósito:

Combina el envío de un comando y la recepción de una respuesta esperada en una sola función para mayor comodidad.

Detalles de Implementación:

- **Envío del Comando:**

  send_command(sock, command)

  - Llama a send_command() para transmitir el comando al controlador del robot.

- **Recepción y Validación de la Respuesta:**

  return recv_response(sock, expected_response)

  - Llama a recv_response() para recibir y validar la respuesta.
  - Devuelve el resultado (True o False) de recv_response().

2.2 Proto-Comandos

Como se discutió anteriormente, los proto-comandos representan acciones de comunicación de nivel intermedio. En esta sección se analizará cómo funcionan estos proto-comandos y cómo utilizan las funciones base. Hemos implementado una secuencia lógica que ejecuta las funciones base de manera escalonada. Las funciones base, además de realizar su acción específica, retornan un valor booleano. Este aspecto es crucial, ya que permite ejecutar múltiples llamadas a funciones base de manera secuencial mediante condicionales simples, asegurando que solo se ejecute una llamada en la secuencia si la llamada anterior se ha completado con éxito.

Vamos a desglosar la función de proto-comando program_select y explicar cómo utiliza las funciones base previamente discutidas (send_command, recv_response y send_and_receive). Exploraremos cómo la lógica escalonada asegura una comunicación robusta con el controlador del robot.

Propósito:

La función program_select es un proto-comando diseñado para seleccionar un programa específico en el controlador del robot. Envía una secuencia de

comandos al controlador para iniciar la selección del programa y confirma que el programa correcto ha sido seleccionado.

Definición de la Función:

```
def program_select(sock, program_name):  # Seleccionar el programa [program_name]
    if send_and_receive(sock, "1", "PROG"):
        if send_and_receive(sock, program_name, program_name):
            time.sleep(1.5)
            return True
    return False
```

Desglose Paso a Paso

1. Iniciar la Selección del Programa

```
if send_and_receive(sock, "1", "PROG"):
    # Continuar si la llamada anterior devuelve True
```

- **Comando Enviado:** "1"
- **Respuesta Esperada:** "PROG"

Explicación:

- **Propósito:** Enviar el comando "1" al controlador del robot indica el inicio del modo de selección de programa.
- **Proceso:**
    - Llama a send_and_receive(sock, "1", "PROG").
    - **En Detalle:**
        - **Envía** el comando "1" usando send_command.
        - **Recibe** y **valida** la respuesta usando recv_response, esperando "PROG".
- **Resultado:**
    - Si la respuesta es "PROG", indica que el controlador está listo para recibir el nombre del programa.
    - Si tiene éxito, la función procede al siguiente paso.
    - Si no, la función devuelve False.

2. Enviar el Nombre del Programa

```
if send_and_receive(sock, program_name, program_name):
    # Continuar si la llamada anterior devuelve True
```

- **Comando Enviado:** program_name (el nombre del programa a seleccionar)
- **Respuesta Esperada:** program_name

Explicación:

- **Propósito:** Envía el nombre del programa a seleccionar al controlador del robot.
- **Proceso:**
    - Llama a send_and_receive(sock, program_name, program_name).
    - **En Detalle:**
        - **Envía** el program_name usando send_command.
        - **Recibe** y **valida** la respuesta usando recv_response, esperando el mismo program_name.
- **Resultado:**
    - Si la respuesta coincide con el program_name, confirma que el controlador ha reconocido la selección del programa.
    - Si tiene éxito, la función procede al siguiente paso.
    - Si no, la función devuelve False.

3. Esperar a que el Programa se Cargue

```
time.sleep(1.5)
return True
```

- **Explicación:**
    - **Propósito:** Introducir un retraso para permitir que el controlador del robot cargue el programa seleccionado.
    - **Proceso:**
        - Llama a time.sleep(1.5) para pausar la ejecución durante 1.5 segundos.
    - **Resultado:**
        - Después del retraso, devuelve True para indicar que el proceso de selección del programa fue exitoso.

4. Devolver False si Algún Paso Falla

```
return False
```

- **Explicación:**
    - Si cualquiera de las llamadas a send_and_receive falla (devuelve False), la función omite los pasos subsecuentes y devuelve False.
    - Esto indica que el proceso de selección del programa no fue exitoso.

2.3 Comandos Complejos

Como se discutió anteriormente, los comandos representan acciones de comunicación de alto nivel. En esta sección se analizará cómo funcionan estos comandos y cómo utilizan los proto-comandos, que a su vez utilizan funciones base. Hemos implementado una secuencia lógica que ejecuta proto-comandos de manera escalonada. Los proto-comandos, además de realizar su acción específica, devuelven un valor booleano. Este aspecto es crucial, ya que permite ejecutar múltiples proto-comandos de manera secuencial mediante condicionales simples, asegurando que solo se ejecute un proto-comando si el anterior se completó con éxito.

Analicemos un comando complejo (num_command_mode == 8) y comprendamos cómo utiliza proto-comandos de manera escalonada para realizar acciones de comunicación de alto nivel con el controlador del robot.

Contexto

Este comando complejo está diseñado para realizar un movimiento completo actualizando las variables correspondientes a cada punto de un programa del robot. Procesa parámetros de entrada, actualiza variables en el controlador del robot e inicia la ejecución del programa seleccionado. El código demuestra cómo se ejecutan múltiples proto-comandos secuencialmente, cada uno dependiendo del éxito del anterior.

Desglose del Código

```
elif num_command_mode == 8:  # Comando para mover completamente aña
diendo un valor a cada punto del programa [Nombre del Programa] [Valor Id 1
]...[Valor Id x]
    correct_change = []
    if (len(parts) - 2) == (len(commands_data["commands"][str(num_command
_mode)][string_command_select])):
        for index, id in enumerate(parts[2:], start=0):
            correct_change.append(False)
            var_value = access_position(commands_data["commands"][str(num_c
ommand_mode)][string_command_select][index], id)
            if var_value:
                if change_variable(sock, commands_data["commands"][str(num_co
```

```
mmand_mode)][string_command_select][index], var_value):
                correct_change[index] = True
        if all(correct_change):  # Verificar que todas las variables se hayan cambi
ado correctamente
            logging.info("Program variables have been set")
            with state_lock:
                condition = state.RESET
            if condition == False:
                reset_prog(sock)
            if servo_on(sock):
                logging.info("Servo Turned On")
                if program_select(sock, string_command_select):
                    logging.info(f"Selected program: {string_command_select}")
                    if run_prog(sock):
                        logging.info("Running")
                        return True
    raise ExecuteError("Incorrect number of variable commands.")
```

Explicación Paso a Paso

1. Inicialización

- **Verificación del Modo de Comando:**

  `elif num_command_mode == 8:`

  - El bloque de código se ejecuta cuando num_command_mode es 8, indicando que se debe ejecutar este comando complejo.

- **Inicializar la Lista correct_change:**

  `correct_change = []`

  - Una lista vacía que rastrea el éxito de los cambios realizados en las variables para cada punto.

2. Validación de Parámetros

- **Verificar la Cantidad de Parámetros:**

  `if (len(parts) - 2) == (len(commands_data["commands"][str(num_comm and_mode)][string_command_select])):`

  - **Propósito:** Asegurarse de que el número de valores proporcionados coincida con el número de variables esperadas para el programa seleccionado.

    - len(parts) - 2: Número de valores proporcionados, excluyendo el modo de comando y el nombre del programa.

        - len(commands_data["commands"][str(num_command_mode)][string_command_select]): Número de variables esperadas para el programa.
    - **Resultado:**
        - Si las cantidades coinciden, el proceso continúa.
        - Si no coinciden, se lanza un ExecuteError al final.

3. Bucle de Actualización de Variables

- **Iterar Sobre los Valores Proporcionados:**

  ```
  for index, id in enumerate(parts[2:], start=0):
  ```

    - **Propósito:** Procesar cada valor proporcionado (id) para las variables del programa.
    - **Proceso:**
        - Itera desde el tercer elemento en parts (omitiendo el modo de comando y el nombre del programa).

- **Intentar Acceso y Actualización de Variables:**

  ```
  correct_change.append(False)
  var_value = access_position(commands_data["commands"][str(num_command_mode)][string_command_select][index], id)
  ```

    - **Acceso a la Posición:**
        - commands_data["commands"][str(num_command_mode)][string_command_select][index] recupera el nombre o identificador de la variable para el índice actual.
        - access_position() es un proto-comando que procesa el identificador de la variable y el valor proporcionado (id), posiblemente realizando validaciones o cálculos.
        - Devuelve var_value si tiene éxito.

- **Actualizar Variable en el Controlador del Robot:**

  ```
  if var_value:
    if change_variable(sock, ..., var_value):
      correct_change[index] = True
  ```

    - **Cambio de Variable:**

- change_variable() es un proto-comando que envía un comando al controlador del robot para actualizar una variable con var_value.
- Si tiene éxito, establece correct_change[index] = True.
- Internamente, se basa en funciones base como send_and_receive().

4. Verificar que Todas las Variables se Han Actualizado Correctamente

- **Comprobar el Éxito de los Cambios en Todas las Variables:**

  ```
  if all(correct_change):
  ```

  - **Propósito:** Asegurarse de que todas las variables se hayan actualizado correctamente antes de continuar.
  - **Resultado:**
    - Si todos los elementos en correct_change son True, el proceso continúa.
    - Si no, la función salta directamente a lanzar un ExecuteError.

5. Secuencia de Ejecución del Programa

- **Registrar Éxito:**

  ```
  logging.info("Program variables have been set")
  ```

- **Reiniciar el Programa si es Necesario:**

  ```
  with state_lock:
    condition = state.RESET
  if condition == False:
    reset_prog(sock)
  ```

  - **Propósito:** Verificar si el programa necesita ser reiniciado.
    - state_lock: Un bloqueo de hilos para sincronizar el acceso al estado compartido.
    - state.RESET: Una bandera que indica si es necesario un reinicio.
  - **Proto-Comando Usado:**
    - reset_prog(sock) reinicia el programa del robot, utilizando internamente funciones base.

- **Encender los Servos:**

```
if servo_on(sock):
    logging.info("Servo Turned On")
```

- **Proto-Comando Usado:**
  - servo_on(sock) envía un comando para activar los motores servo del robot.

- **Seleccionar el Programa:**

```
if program_select(sock, string_command_select):
    logging.info(f"Selected program: {string_command_select}")
```

- **Proto-Comando Usado:**
  - program_select(sock, string_command_select) selecciona el programa especificado en el controlador del robot.

- **Ejecutar el Programa:**

```
if run_prog(sock):
    logging.info("Running")
    return True
```

- **Proto-Comando Usado:**
  - run_prog(sock) inicia la ejecución del programa seleccionado.
- **Éxito:**
  - Si todos los pasos anteriores son exitosos, la función devuelve True.

6. Manejo de Errores

- **Error de Desajuste de Parámetros:**

```
raise ExecuteError("Incorrect number of variable commands.")
```

  - **Propósito:** Lanza una excepción si el número de parámetros proporcionados no coincide con la cantidad esperada.
  - **Resultado:** Señala al código que llama que la ejecución del comando falló debido a una entrada incorrecta.

Cómo se Utilizan los Proto-Comandos

El comando complejo se basa en varios proto-comandos para realizar sus operaciones, cada uno construido sobre funciones base:

1. **access_position(variable_name, id)**

- Procesa y valida el valor de la variable para el identificador dado.
- Puede involucrar cálculos o recuperación de datos.
- Utiliza funciones base para cualquier cálculo necesario.

2. **change_variable(sock, variable_name, var_value)**
   - Actualiza una variable en el controlador del robot con el nuevo valor.
   - Internamente utiliza funciones base como send_and_receive() para comunicarse con el controlador.
   - Devuelve un valor booleano que indica éxito.
3. **reset_prog(sock)**
   - Reinicia el programa del robot a un estado conocido.
   - Asegura que el controlador esté listo para ejecutar un nuevo programa.
   - Utiliza funciones base para la comunicación.
4. **servo_on(sock)**
   - Activa los motores servo del robot.
   - Es necesario antes de ejecutar un programa.
   - Utiliza funciones base para enviar el comando correspondiente.
5. **program_select(sock, string_command_select)**
   - Selecciona el programa especificado en el controlador del robot.
   - Como se analizó previamente, utiliza send_and_receive() para comunicarse.
6. **run_prog(sock)**
   - Inicia la ejecución del programa seleccionado.
   - Utiliza funciones base para enviar el comando de ejecución y confirmar la ejecución.

2.4 Definición de Comandos

El archivo **commands.json** sirve como una biblioteca de comandos predefinidos disponibles para el sistema. Contiene diferentes categorías de comandos, como se muestra a continuación:

```
{
  "commands": {
    "1": {
      "START": {},
      "ERROR": {},
      "PRUEBA": {},
      "PRCHAVAR": {}
    },
    "2": {},
    "3": {},
    "4": {},
    "5": {},
    "6": {
      "P_00": {},
      "P_01": {},
      "P_02": {},
      "P_03": {},
      "P_04": {},
      "J_00": {},
      "J_01": {},
      "J_02": {},
      "J_03": {},
      "J_04": {},
      "M_00": {},
      "M_01": {},
      "M_02": {},
      "M_03": {},
      "M_04": {},
      "D_00": {},
      "D_01": {},
      "D_02": {},
      "D_03": {},
      "D_04": {}
    },
    "7": {},
    "8": {
      "PRCHAVAR": ["P_00"],
      "PRDIF": ["P_00", "P_01", "J_00", "M_00", "C_00"],
      "MOVPOT": ["P_00", "P_01", "P_02", "C_00"],
      "PLANTSEED": ["P_03", "P_00"],
      "TAKEPHOTO": ["P_00"]
    }
```

```
  }
}
```

- **Comandos Básicos**: Enumerados en categorías (por ejemplo, “2” a “5” y “7”) y representan operaciones básicas. Estos comandos no requieren argumentos adicionales. Ejemplos de estos comandos son:

  2. Reanudar
  3. Parada Temporal
  4. Detener y Reiniciar
  5. Reiniciar Errores
  7. Mostrar Estado

- **Comandos Parametrizados**: Comandos más complejos como “1”, “6” u “8” que requieren uno o más argumentos:

  1. Ejecutar un programa. Arg: [Nombre del Programa]. Este comando encenderá los servos, verificará si el slot 1 ya está reiniciado, si no, lo reiniciará, seleccionará el programa para cargar en el slot 1 y lo ejecutará. Para esta secuencia, los mensajes enviados al controlador serían 6, 4 (si es necesario), 1, [Nombre del Programa], 2.
  6. Modificar una variable. Arg: [Nombre de la Variable], [ID del Valor]. Este comando verifica si la variable existe en el archivo commands.json, luego envía el comando 9 para ingresar al estado de modificación de valores de la variable y cambia el valor basado en point_data.json.
  8. Programas para controlar el robot con argumentos. Args: [Nombre del Programa] [ID del Valor 1] ... [ID del Valor n]. Este comando ejecuta un programa que requiere datos de entrada, como puntos objetivo, selección de modo, etc. Combina los procesos de los comandos 1 y 6.

2.5 Manejo de Argumentos

Los argumentos requeridos para estos comandos están definidos en el archivo **point_data.json**, donde se almacenan posiciones específicas, valores de juntas y constantes numéricas o de texto.

- **Posiciones y Datos de Juntas**: Por ejemplo, la sección "positions" proporciona posiciones precisas del robot en coordenadas cartesianas o ángulos de juntas. Estos datos son referenciados por comandos que requieren movimientos específicos:

```
"positions": {
  "Joint": {
    "1": "(-28,+21,+93,+0,+90,+0)",
    "2": "(+7,+8,+9,+10,+11,+12)"
  },
  "Point": {
    "POT1": "(+400.00,+300.00,+300.00,+180.00,+0.00,+90.00,+0.00
,+0.00)"
  }
}
```


Cuando se invoca un comando como PRDIF, los argumentos enumerados (P_00, P_01, J_00, etc.) se recuperan de este archivo, y los valores correspondientes se sustituyen en el comando antes de enviarlo al robot.

- **Datos de Texto y Numéricos**: El archivo **point_data.json** también contiene cadenas simbólicas y constantes numéricas utilizadas para tareas específicas (por ejemplo, IN, OUT, +1, +0). Estos pueden ser usados en comandos que requieren toma de decisiones o acciones binarias (como activar o desactivar una función).

3. Flujo de Trabajo Ejemplo: Ejecución de un Comando Personalizado

1. **Entrada del Usuario**: Un cliente envía una solicitud al servidor FastAPI para intercambiar la posición de POT1 y POT2 utilizando DESK1 como referencia.
   El usuario invoca el comando MOVPOT con argumentos de puntos específicos, siguiendo la estructura definida en commands.json:


   ```
   "MOVPOT": ["P_00", "P_01", "P_02", "C_00"],
   ```


   Un ejemplo de invocación de este comando podría ser:

   8 MOVPOT POT1 DESK1 POT2 CHANGE

2. **Búsqueda del Comando**: El servidor verifica en commands.json que MOVPOT es un comando válido y que acepta los argumentos proporcionados.

3. **Resolución de Argumentos**: El servidor recupera los datos de posición para POT1, DESK1, POT2 y CHANGE desde point_data.json:


   ```
   "POT1": "(+400.00,+300.00,+300.00,+180.00,+0.00,+90.00,+0.00,+
   0.00)"
   ```


4. **Formateo del Comando**: El comando es formateado por el servidor, incorporando los datos de posición resueltos.

5. **Transmisión del Comando**: El comando formateado es enviado al robot para su ejecución, y los argumentos se escriben en las variables globales establecidas: P_00, P_01, P_02 y C_00.

6. **Manejo de Respuesta**: El servidor espera una respuesta del robot y retransmite el resultado (éxito o error) al cliente.

4. Diagrama de los Comandos de Alto Nivel

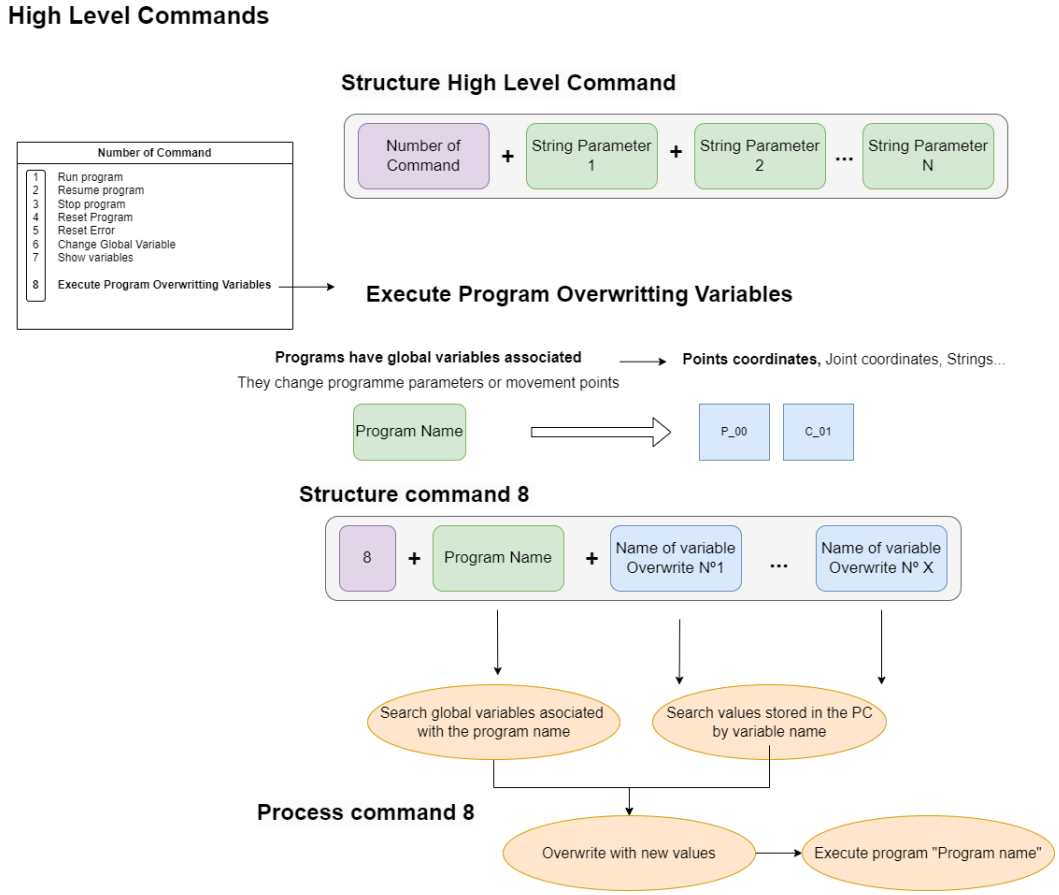

Ilustración 72: Diagrama que muestra cómo se ejecuta el comando complejo 8 para el paso de parámetros

En la Ilustración 72 se muestra cómo se ejecutan los comandos complejos 8, para ejecutar programas que necesitan argumentos.

Dichos argumentos se le pasan al programa seleccionado sobrescribiendo sus variables a través de mensajes en el canal de comunicación

5. Conclusión

El sistema de gestión de comunicación y comandos en esta interfaz FastAPI con el robot Mitsubishi Melfa está estructurado para facilitar tanto operaciones simples como complejas. Al utilizar plantillas de comandos predefinidas y un enfoque basado en JSON para la gestión de parámetros, el sistema puede

manejar de manera eficiente comandos definidos por el usuario que requieren argumentos dinámicos.

5.5 Gemelo Digital

Dado a la gran utilidad de un gemelo digital para realizar pruebas y simulaciones de una forma segura y que además permite la inclusión en los proyectos de trabajadores de una forma remota, decidimos investigar como realizar un gemelo digital, que no solo nos permitiera realizar simulaciones, sino que también nos permitiera visualizar en tiempo real los movimientos reales del robot, característica útil para trabajadores en remoto.

Para conseguir la capacidad de 'espejo' del robot fue sumamente importante el método de comunicación escogido y diseñado, ya que permite directamente enviar la variable interna del robot que almacena la posicion actual del robot, tal y como predijimos al realizar el análisis de las distintas opciones de comunicación. Si hubiéramos elegido un método clásico de comunicación con el controlador, hubiéramos necesitado 32 señales digitales, ya que es el tamaño de la información que contiene la variable interna que almacena la posicion del robot.

5.5.1 Introducción

Este apartado proporciona una guía completa para implementar un **Gemelo Digital** de un robot Mitsubishi utilizando la plataforma **ISG-Virtuos**. El gemelo digital tiene dos características principales: refleja los movimientos del robot físico interactuando en un entorno virtual 3D y puede simular el robot de manera independiente, garantizando la monitorización y prueba en tiempo real.

En el momento en el que decidimos implementar un Gemelo Digital, aunque fue sencillo realizar la funcionalidad de 'espejo' gracias al protocolo de comunicación realizado, teníamos que encontrar también la manera de realizar simulaciones sin el robot real. Esto era muy tedioso, ya que, sin el software oficial de Mitsubishi, no disponíamos de un compilador que pudiera ejecutar los programas en Melfa Basic-V. Por lo tanto, la empresa consiguió darnos acceso en a un ordenador con una licencia de dicho software, llamado RT Toolbox, gracias al cual Podemos compilar y ejecutar programas en el lenguaje de programación del robot.

Los componentes clave incluyen:

- Configuración del RT Toolbox de Mitsubishi y del Controlador CR750D para la comunicación.
- Programa en MELFA Basic V para transmitir continuamente las posiciones de las articulaciones.

- Configuración del espacio de trabajo de ISG-Virtuos para la sincronización.
- Scripts en Python para el intercambio de datos en tiempo real.

Esta guía asegura un flujo de trabajo fluido entre el robot físico y su contraparte digital.

5.5.2 Configuración de RT Toolbox

El **RT Toolbox** es un software de configuración proporcionado por Mitsubishi Electric para gestionar y programar robots industriales. En este proyecto, el RT Toolbox se utiliza para simular los programas del robot y enviar las posiciones de las articulaciones a ISG-Virtuos. Las siguientes secciones documentan la configuración del RT Toolbox para direcciones IP y ajustes de dispositivos y líneas, con el objetivo de garantizar una integración fluida entre el robot físico y el entorno de simulación digital.

- **Acceder a la Configuración de Ethernet:**
    - Navegar en el menú izquierdo de la interfaz de RT Toolbox -> 'Tu nombre de proyecto'/ Simulación / Parámetro / Parámetro de Comunicación / Ethernet.

Configuración de Dirección IP

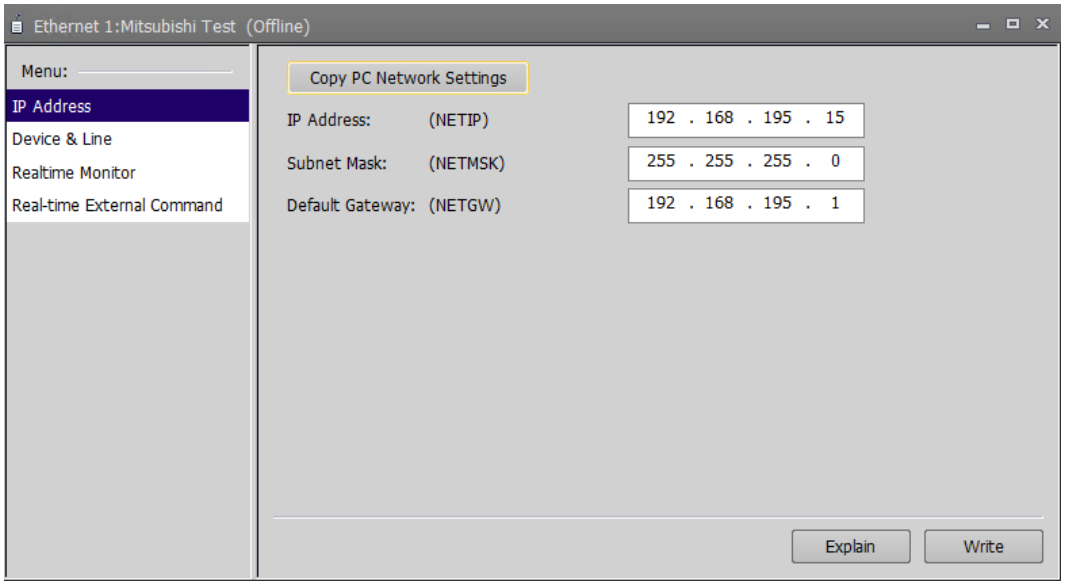

Ilustración 73: Configuración de la IP de Ethernet1 en ISG Virtuos

- **Ajustes:**
    - **Dirección IP:** 192.168.195.15
    - **Máscara de Subred:** 255.255.255.0

- **Puerta de Enlace Predeterminada:** 192.168.195.1

Estos ajustes (Ilustración 73) establecen la conexión de red del controlador del robot. Es importante escribir la dirección IP del ordenador que está simulando el robot en el campo **Dirección IP**.

Configuración de Dispositivo y Línea

La configuración de dispositivo-línea asigna dispositivos específicos y sus ajustes de comunicación dentro del RT Toolbox.

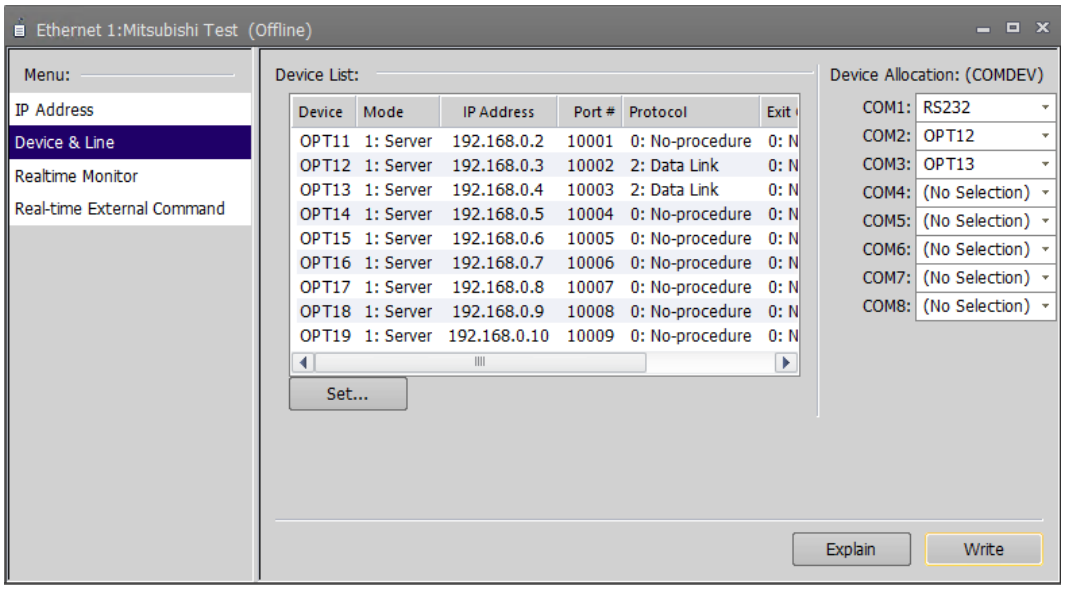

Ilustración 74: Configuración de dispositivo y línea del Ethernet1 en ISG Virtuos

- **Lista de Dispositivos:**
  - Cada dispositivo (por ejemplo, OPT11, OPT12) está asignado a una dirección IP y un puerto para la comunicación.
  - Protocolo: El protocolo de comunicación debe establecerse como Data Link.
- **Asignación de Dispositivo (COMDEV):**
  - Los puertos COM en el PC host están vinculados a dispositivos específicos.
  - Es importante configurar **COM3 como OPT13**, ya que más adelante utilizaremos COM3 para enviar los datos de posición.

Estas configuraciones (Ilustración 74) forman la base para una comunicación estable entre el controlador y ISG-Virtuos.

Nota: La configuración de **COM2 y OPT12** se utiliza para recibir el estado del robot en la Raspberry y el Front-End, pero esto no es necesario para el gemelo digital.

5.5.3 Programa en MELFA Basic V

Este programa, escrito en MELFA Basic V, está diseñado para ejecutarse continuamente en una de las ranuras del controlador. Su propósito principal es mantener un canal de comunicación abierto a través de una línea Ethernet (COM3) y enviar periódicamente las posiciones actuales de las articulaciones del robot. A continuación, se detalla el código:

Desglose del Código

```
Open "COM3:" As #2 'Abrir línea Ethernet
*R
Print #2, J_Curr
GoTo *R
End
```

1. Abre COM3 para la comunicación a través de Ethernet.
2. Transmite J_Curr, que contiene las posiciones actuales de las articulaciones, al sistema conectado.
3. Se ejecuta en un bucle infinito para garantizar una transmisión continua de datos.

Caso de Uso en el Gemelo Digital

Los datos de posición de las articulaciones enviados por este programa simple son consumidos por la simulación en ISG-Virtuos, asegurando que el gemelo digital refleje en tiempo real los movimientos físicos del robot.

5.5.4 Configuración del Espacio de Trabajo en ISG-Virtuos

Descripción General de la Estación 3D

El espacio de trabajo de ISG-Virtuos es el entorno donde se simula el gemelo digital del robot Mitsubishi y su estación 3D correspondiente. Esta documentación explica los componentes clave de la configuración, incluyendo la configuración de la estación 3D, la comunicación TCP/IP, el diagrama de bloques para la cinemática del robot y el módulo de conversión de grados a radianes.

La estación 3D replica el entorno del robot, incluyendo: - El robot, herramientas y accesorios. - Bancos de trabajo para la ejecución de tareas.

Esta configuración virtual permite validar procesos sin interrumpir las operaciones físicas.

Modelo 3D y Espacio de Trabajo

Los modelos CAD 3D pueden ser incluidos en el espacio de trabajo de ISG. Además, ISG proporciona herramientas de modelado 3D y configuración lógica.

- **Modelo 3D** (Ilustración 75):

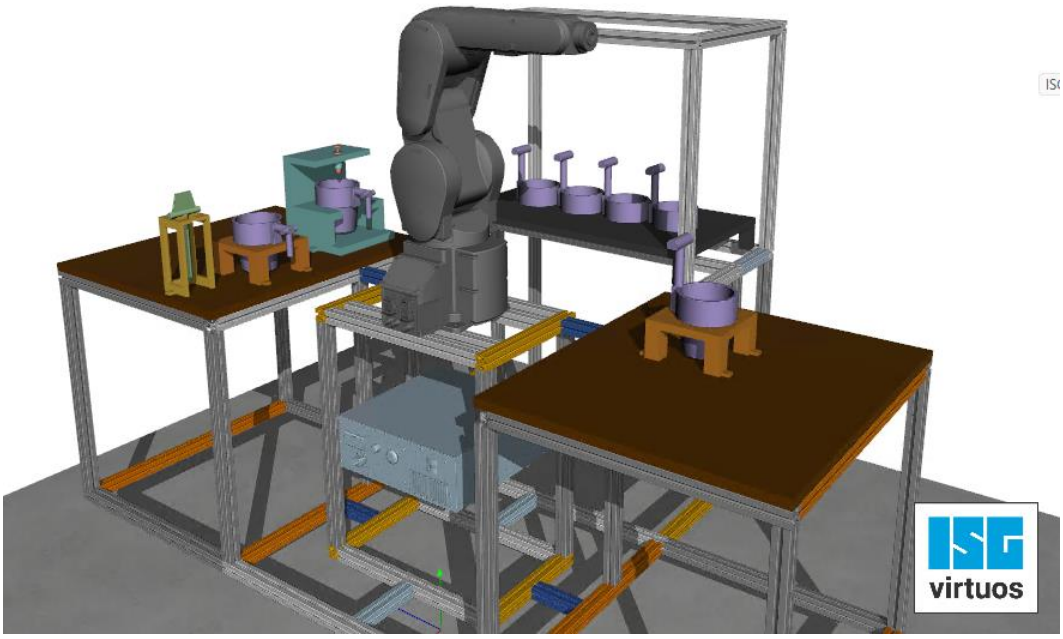

Ilustración 75: Modelo 3D desfasado que se incluyó en ISG Virtuos para las primeras pruebas

- **Espacio de Trabajo en ISG** (Ilustración 76):

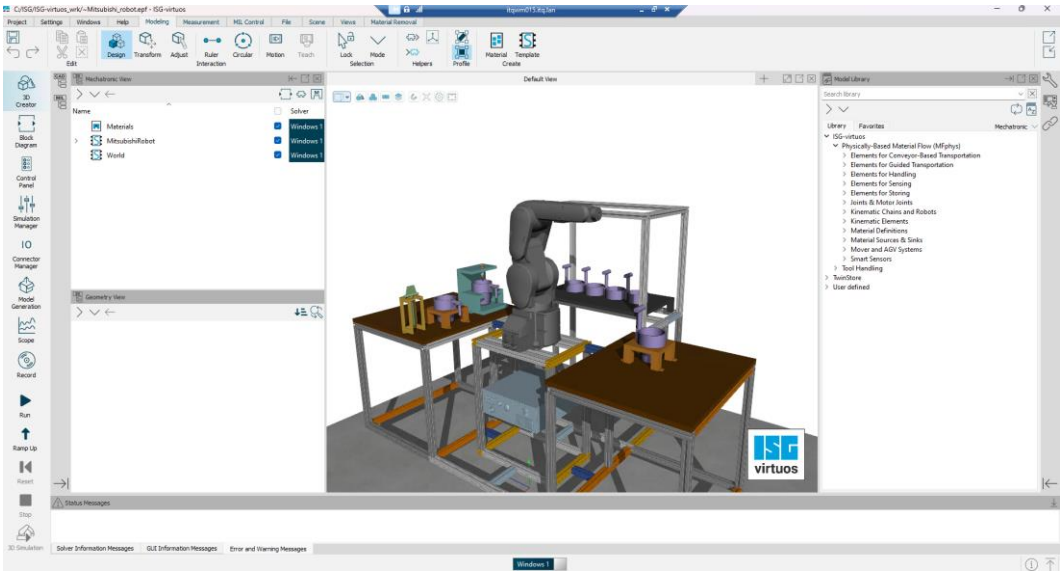

Ilustración 76: Espacio de trabajo de ISG Virtuos

Configuración del Bloque de Conexión TCP/IP

Este bloque configura la comunicación entre el robot y ISG-Virtuos utilizando el protocolo **TCP/IP** (Ilustración 77).

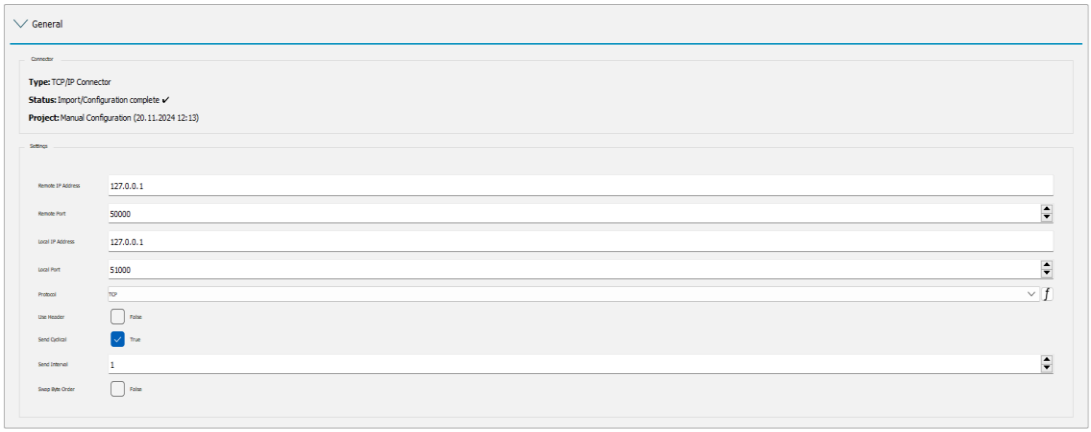

Ilustración 77: Configuración del bloque TCP/IP

- **Ajustes del Conector:**
  - **IP Remota:** 127.0.0.1
  - **Puerto Remoto:** 50000
  - **IP Local:** 127.0.0.1
  - **Puerto Local:** 51000
  - **Usar encabezado:** False

Los datos se envían de forma cíclica (cada segundo) entre el robot y ISG-Virtuos, permitiendo actualizaciones en tiempo real.

Diagrama de Bloques para el Robot Mitsubishi

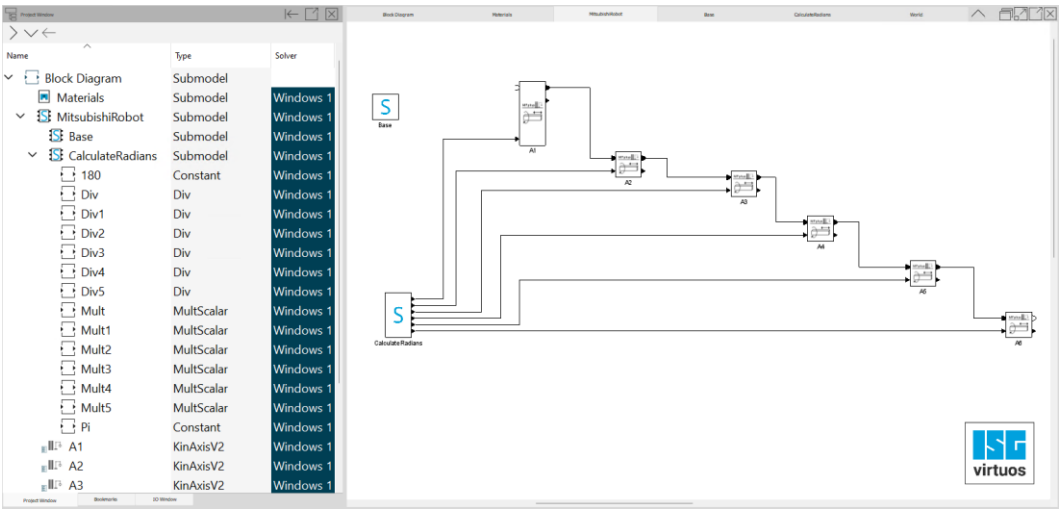

Ilustración 78: Diagrama de bloques que modela al robot

Este diagrama de bloques (Ilustración 78) modela la cinemática del robot Mitsubishi e integra su funcionamiento en la simulación:

- **Bloque Base:** Representa el marco de referencia de origen del robot.

- **Bloques de Ejes (A1–A6):** Reflejan el movimiento de cada articulación utilizando los datos recibidos.
- **Conexiones de Entrada:** Reciben los datos (por ejemplo, posiciones articulares) desde la interfaz TCP/IP para controlar la simulación (estos datos entran en el bloque de conversión).

El siguiente bloque, mostrado en la Ilustración 79 convierte los ángulos de las articulaciones de grados (utilizados en la programación del robot) a radianes (utilizados internamente en muchos cálculos matemáticos).

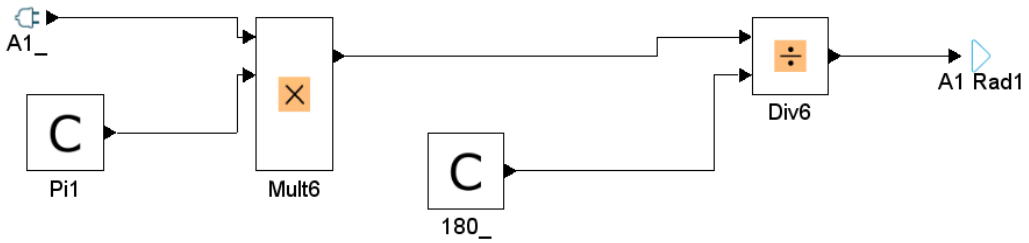

Ilustración 79: Bloque transformador de ángulos de grados a radianes

- **Entrada:** Ángulos de articulaciones en grados.
- **Fórmula:**

$$\text{radianes} = \text{grados} \times \frac{\pi}{180}$$

5.5.5 Controlador de Python para Comunicación de Datos

Para facilitar la comunicación entre el controlador del robot Mitsubishi y el gemelo digital ISG-Virtuos, se utiliza un controlador en Python que se ejecuta en el backend:

- sendPosition.py: Maneja la recepción y transmisión en tiempo real de los datos de los ángulos de las articulaciones entre el controlador y el gemelo digital.

Explicación del Código de sendPosition.py

Este fichero opera en segundo plano y automatiza el proceso de recuperación de los ángulos de las articulaciones del controlador del robot Mitsubishi y su envío a ISG-Virtuos para la simulación en tiempo real.

- **Funcionalidad:**
    - Establece conexiones TCP persistentes tanto con el controlador como con el gemelo digital.

- Recibe los datos de los ángulos de las articulaciones del controlador de forma asíncrona.
- Convierte los ángulos de las articulaciones en formato binario para su transmisión.
- Envía los datos procesados a ISG-Virtuos mediante una conexión de socket.
- Maneja fallos de conexión e intenta reconexiones automáticas.

Componentes Clave

1. **Configuración de Socket:**

```
CONTROLLER_IP = '192.168.0.20'  # Dirección IP del controlador del robot
CONTROLLER_PORT = 10003  # Puerto para la comunicación TCP/IP
OUTPUT_HOST = '192.168.195.15'  # Dirección del socket del gemelo digital
OUTPUT_PORT = 51000  # Puerto del socket del gemelo digital
```

   - **Socket del Controlador:** Se conecta al controlador del robot Mitsubishi para recuperar los datos de los ángulos de las articulaciones.
   - **Socket de Salida (Gemelo Digital):** Envía los datos de las articulaciones en formato binario a ISG-Virtuos.

2. **Manejo de Conexiones Asíncronas:**

```
async def connect_to_controller(controllerHost, controllerPort):
    while True:
        try:
            controller_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            controller_socket.connect((controllerHost, controllerPort))
            return controller_socket
        except Exception as e:
            await asyncio.sleep(1)  # Reintentar después de un retraso
```

   - Intenta continuamente establecer una conexión con el controlador del robot.
   - Implementa lógica de reintento para manejar fallos de conexión.

3. **Recepción y Procesamiento de Datos en Tiempo Real:**

```
async def pos_socket_reader(controllerHost, controllerPort, bufferSize, readInterval, position_data_queue):
```

```
    controller_socket = await connect_to_controller(controllerHost, contr
olerPort)
    while True:
        try:
            data = controller_socket.recv(bufferSize).decode('utf-8').strip()
            if data:
                await position_data_queue.put(data)  # Almacenar datos en
una cola
        except Exception:
            controller_socket.close()
            controller_socket = await connect_to_controller(controllerHost,
controllerPort)  # Reconectar
```

- Lee los datos de los ángulos de las articulaciones del controlador.
- Almacena los datos recibidos en una cola asíncrona para su procesamiento.
- Implementa manejo de errores y reconexión automática en caso de fallos.

4. **Procesamiento de Datos y Transmisión a ISG-Virtuos:**

```
async def sim_data_processor(outputHost, outputPort, readInterval, po
sition_data_queue):
    output_socket = await connect_to_output(outputHost, outputPort)
    while True:
        if not position_data_queue.empty():
            data = await position_data_queue.get()
            values = parse_angles_to_floats(data)
            if values:
                binary_data = encode_floats_to_binary(values)
                output_socket.sendall(binary_data)
```

- Recupera los datos de la cola, los procesa y los envía a ISG-Virtuos.
- Garantiza un flujo continuo de datos mientras mantiene conexiones estables.

5. **Análisis y Codificación de Datos:**

```
def parse_angles_to_floats(angle_string):
    angles = angle_string.strip("()").split(",")
    return [float(angle) for angle in angles]
```

```
def encode_floats_to_binary(float_values):
    return struct.pack('6f', *float_values)
```

- Convierte una cadena de ángulos de las articulaciones en una lista de valores flotantes.
- Codifica la lista de valores flotantes en un formato binario (floats de 32 bits) para una transmisión eficiente.

6. **Gestión del Estado de Conexión:**

```
async def update_connection_state(setConnection):
    url = "http://127.0.0.1:8000/api/set-isg-connection"
    payload = {"setConnection": setConnection}
    requests.post(url, json=payload)
```

- Actualiza el estado del sistema respecto a la conexión con el entorno ISG-Virtuos.
- Ayuda a monitorear el estado del flujo de datos en tiempo real.

Resumen del Controlador en Python

Tabla 10: Resumen del fichero sendPosition.py

Fichero	Propósito	Características Clave
sendPosition.py	Transmisión de datos de articulaciones en tiempo real	Procesamiento asíncrono, reconexiones automáticas, codificación binaria

En la tabla 10 se puede ver el resumen del fichero encargado de enviar la posición del robot en tiempo real.

Este controlador en el backend proporciona un marco robusto y eficiente para garantizar la integración en tiempo real entre el robot Mitsubishi y ISG-Virtuos, manejando posibles fallos de conexión mientras mantiene un flujo de datos continuo.

5.5.6 Imagen de la Simulación

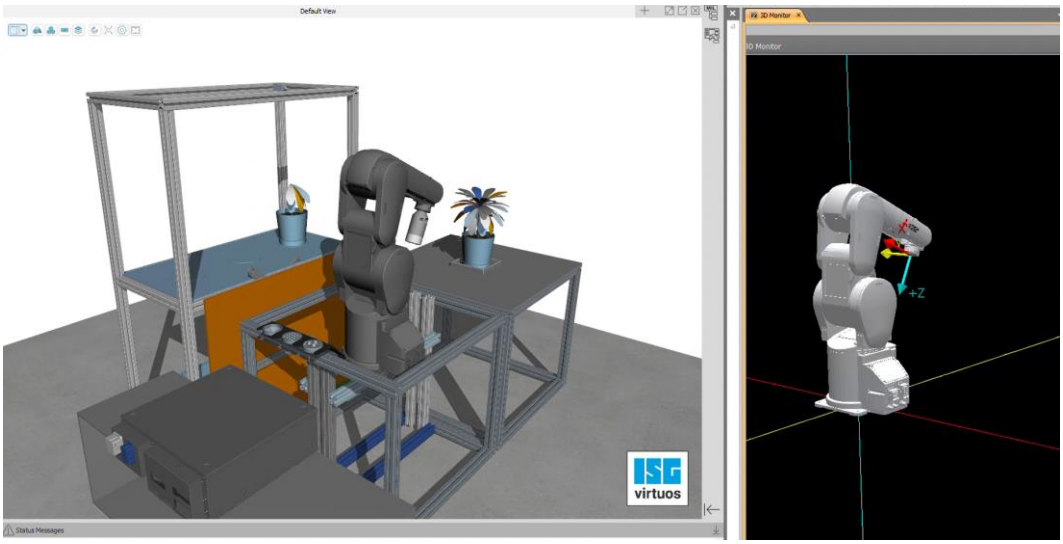

Ilustración 80: Funcionalidad de simulación del gemelo digital

En la Ilustración 80 se puede ver la funcionalidad de simulación del Gemelo Digital

5.5.7 Imagen de la Funcionalidad Espejo

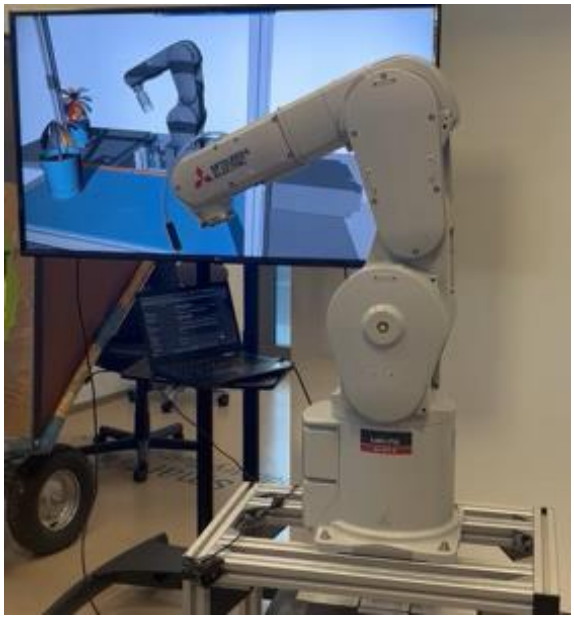

Ilustración 81: Funcionalidad de espejo del gemelo digital

En la Ilustración 81 se puede ver la funcionalidad de espejo del gemelo digital

5.5.8 Conclusión

Esta documentación describe el proceso completo de creación de un gemelo digital para un robot Mitsubishi utilizando ISG-Virtuos. Al integrar hardware,

software de simulación y programas personalizados en Python, el proyecto logra:

- Sincronización en tiempo real entre los sistemas físicos y virtuales.
- Un marco sólido para la prueba y validación de tareas robóticas.

Esta solución demuestra la efectividad de la comunicación TCP/IP a través de Ethernet. Se logró fácilmente un gemelo digital que mejora la eficiencia, reduce el tiempo de inactividad y garantiza simulaciones de procesos precisas.

6 Análisis del Estado de Imágenes de Plantas Usando Inteligencias Artificiales LLM Multimodales

Un objetivo del proyecto era reducir al máximo las acciones del agricultor durante el proceso de germinación de las semillas y automatizar al máximo los procesos y la toma de decisiones. Para ello, se fijó como objetivo automatizar el reconocimiento del estado de crecimiento de las plantas.

Una vez fijado el objetivo comenzamos la investigación de posibles herramientas y soluciones para llevar esto a cabo. Nuestra primera idea fue desarrollar nuestro propio modelo de visión artificial, pero los hechos de que para ello es necesario entrenarlo con un amplio banco de datos y que se necesita invertir una gran cantidad de tiempo y trabajo en ello limitaban la viabilidad de incluir esta funcionalidad en el proyecto.

Sin embargo, tras seguir investigando opciones vimos que era posible utilizar un Large Language Model multimodal para automatizar esta tarea. Encontramos dos videos explicativos que nos sirvieron de inspiración: **[17]** y **[18],** y nos pusimos a desarrollar las primeras pruebas de concepto, para demostrar la viabilidad de esta funcionalidad, usando para ello la documentación oficial de Open AI **[19].**

Una vez visto que era posible obtener información estructurada de la API de GPT y que dicha información podía ser fácilmente integrada en el sistema para la toma de decisiones solo quedaba probar si este método era lo suficientemente determinista y preciso para ser utilizado para la toma de decisiones en un sistema ciber físico real. Para ello se realizaron pruebas exhaustivas de calidad y repetibilidad y un ajuste fino continuo de la ingeniería del prompt.

En este capítulo se explicará todo el proceso de desarrollo de la funcionalidad de análisis del estado de plantas a través de imágenes. Se explica cómo usar las diferentes herramientas, se incluyen todos los análisis relevantes, el proceso de pruebas y perfeccionamiento, y la ingeniería de prompt.

No solo se analiza la funcionalidad final seleccionada, una API GPT usando el modelo GPT-4o, si no que en el Anejo 3 también se muestra cómo se han analizado, probado y comparado otros modelos LLM multimodales para seleccionar el óptimo para nuestro caso de uso.

Finalmente se utiliza la API de OpenAI GPT para analizar imágenes de plantas y proporcionar datos estructurados sobre su germinación, crecimiento y posibles enfermedades. Esta capitulo describe el proceso de manejo de

imágenes, comunicación con la API, predicciones de costes y resultados de pruebas.

6.1 Descripción General

El objetivo de este proyecto es desarrollar un proceso automatizado que analice imágenes de plantas para determinar:

- Si la planta ha germinado.
- Si la planta ha crecido lo suficiente como para ser retirada del módulo de germinación.
- Si la planta presenta enfermedades y, en caso afirmativo, identificar la enfermedad específica.

Se utiliza la API de GPT (específicamente modelos como GPT-4o y GPT-4o-mini) para analizar imágenes y generar respuestas estructuradas en formato JSON. Esta documentación detalla la implementación en Python, los resultados de las pruebas, predicciones de costes y posibles mejoras al sistema.

6.2 Implementación

6.2.1 Importaciones

```
from pydantic import BaseModel
import base64
import requests
import json
```

- **pydantic.BaseModel**: Se utiliza para definir y validar la estructura de la respuesta de la API.
- **base64**: Codifica datos binarios, como imágenes, en formato base64 para su transmisión en la solicitud a la API.
- **requests**: Permite realizar solicitudes HTTP POST a la API de OpenAI.
- **json**: Maneja datos JSON para enviar cargas útiles y procesar respuestas.

6.2.2 Clave de API

```
# OpenAI API Key (It is important to keep API keys private and avoid hardcodin
g them in production)
with open("/run/secrets/api_key_gpt") as f:
    api_key = f.read().strip()
    logging.info(api_key)
```

- **api_key**: Clave de la API de OpenAI utilizada para la autenticación. Debe almacenarse de manera segura y no debe incluirse directamente en entornos de producción.

6.2.3 Definición del Modelo Pydantic

```
class PlantState(BaseModel):
    germinated: bool
    grown: bool
    disease: bool
    whatdisease: str
```

- **PlantState**: Un modelo Pydantic para garantizar que la respuesta de la API cumpla con la estructura esperada.
    - **germinated**: Booleano que indica si la planta ha germinado.
    - **grown**: Booleano que indica si la planta ha crecido lo suficiente.
    - **disease**: Booleano que indica si la planta tiene alguna enfermedad.
    - **whatdisease**: Especifica el nombre de la enfermedad si la planta está infectada; de lo contrario, “none”.

6.2.4 Función para Codificar Imágenes

```
def encode_image(image_path):
    with open(image_path, "rb") as image_file:
        return base64.b64encode(image_file.read()).decode('utf-8')
```

- **encode_image(image_path)**: Esta función toma la ruta de la imagen y la codifica en una cadena base64 para su transmisión en la solicitud a la API.

6.2.5 Ruta de Imagen y Conversión a Base64

```
image_path = "path_to_your_image.jpg"
base64_image = encode_image(image_path)
```

- **image_path**: Ruta del archivo de la imagen de la planta que se analizará.
- **base64_image**: La imagen se convierte en una cadena base64 utilizando la función encode_image.

6.2.6 Configuración de la Solicitud a la API

Encabezados

```
headers = {
    "Content-Type": "application/json",
```

```
    "Authorization": f"Bearer {api_key}"
}
```

- **Content-Type**: Especifica que los datos enviados están en formato JSON.
- **Authorization**: Incluye la clave de la API mediante autenticación Bearer.

Carga Útil

```
payload = {
    "model": "gpt-4o",
    "messages": [
        {
            "role": "system",
            "content": [
                {
                    "type": "text",
                    "text": "You are an expert plant care-taker system..."
                }
            ]
        },
        {
            "role": "user",
            "content": [
                {
                    "type": "text",
                    "text": "Extract the state of the plant from an image..."
                },
                {
                    "type": "image_url",
                    "image_url": {
                        "url": f"{base64_image}",
                        "detail": "high"
                    }
                }
            ]
        }
    ],
    "max_tokens": 200
}
```

- **model**: Especifica el modelo GPT a utilizar, como "gpt-4o" o "gpt-4o-mini".
- **messages**: Una conversación estructurada donde el system define la tarea para la IA y el user proporciona la imagen de la planta para análisis.

- **max_tokens**: Limita el uso de tokens en la respuesta a 200.

6.2.7 Envío de la Solicitud a la API

response = requests.post("https://api.openai.com/v1/chat/completions", headers=headers, json=payload)

- **requests.post()**: Envía una solicitud POST a la API de OpenAI con los encabezados y la carga útil especificados.

6.2.8 Procesamiento de la Respuesta de la API

```
# Function to check and process the response
async def check_response(response):
    if response.status_code == 200:
        try:
            # Parse the response JSON
            response_data = response.json()
            # Extract the content message containing the structured response (in JSON format)
            message_content = response_data["choices"][0]["message"]["content"]

            # If the response contains the error message "no plant or pot has been detected", handle it
            if "no plant or pot has been detected" in message_content.lower():
                error_response = {"Error": "No plant or pot has been detected."}
                # Print the token usage from the response, if available
                print(f"The token usage is: {response_data.get('usage', 'No token usage information available')}")
                return json.dumps(error_response)

            else:
                # Clean the response content and convert it into a Python dictionary
                plant_data = json.loads(message_content.strip("```json\n").strip("\n```"))

                # Validate the JSON content using the Pydantic model (PlantState)
                plant_state = PlantState(**plant_data[0])
                        # Print the token usage from the response, if available
                print(f"The token usage is: {response_data.get('usage', 'No token usage information available')}")
                # Return the validated response as JSON
                return plant_state.json()
```

```
        except Exception as e:
            # Return a JSON with an error message if something goes wrong
            await handlerChatGptError(f"Error of ChatGPT API processing the respo
nse: {str(e)}")
            error_response = {"Error": f"Error of ChatGPT API processing the respon
se: {str(e)}"}
            return json.dumps(error_response)

    else:
        await handlerChatGptError(f"Error of ChatGPT API request failed with stat
us code: {response.status_code}, response: {response.text}")
        # Return a JSON with an error message for a failed API request
        error_response = {"Error": f"Error of ChatGPT API request failed with statu
s code: {response.status_code}, response: {response.text}"}
        return json.dumps(error_response)
```

- **Caso de Éxito**:
    - Analiza la respuesta JSON, extrae el contenido estructurado y lo valida utilizando el modelo PlantState.
    - Si la imagen contiene una planta, imprime los datos del estado de la planta y el uso de tokens.
- **Manejo de Errores**:
    - **Caso de Imagen Irrelevante**: Si la API detecta que la imagen no contiene una planta o maceta (por ejemplo, una foto de una persona), la respuesta incluirá el mensaje "no plant or pot has been detected". En este caso, el sistema imprime este mensaje de error y no procede con la validación del estado de la planta.
    - **Manejo de Excepciones**: Si ocurre un problema al procesar la respuesta (por ejemplo, si el formato JSON es inválido o no se puede analizar), se lanza una excepción y se muestra un mensaje de error.
    - **Fallo de la API**: Si la solicitud a la API falla (por ejemplo, debido a un problema de red o una solicitud inválida), se muestra un mensaje de error con el código de estado y el contenido de la respuesta.

6.3 Predicciones de Tokens y Costes

- Antes de realizar las pruebas, se intentó predecir el coste base de cada solicitud, determinado por la imagen enviada y el prompt utilizado en cada petición.

- Como el prompt será siempre el mismo, se desarrolló un script en Python para predecir el consumo de tokens.
- Para la predicción del coste del procesamiento de imágenes, se utilizó una calculadora oficial.

6.3.1 Precios de los Modelos

Model	Input price for 1M tokens	Output price for 1M tokens	Price per API call
gpt-4o	$5.00	$15.00	$0.0113
gpt-4o-mini	$0.15	$0.60	$0.0004
gpt-4-turbo	$10.00	$30.00	$0.0226
gpt-4	$30.00	$60.00	$0.0529
gpt-3.5-turbo	$0.50	$1.50	$0.0011

Ilustración 82: Comparación de los precios de los distintos modelos de gpt

La Ilustración 82 muestra una comparación de los precios de cada modelo de gpt disponible.

6.3.2 Consumo de Tokens para el Procesamiento de Imágenes con GPT-4o

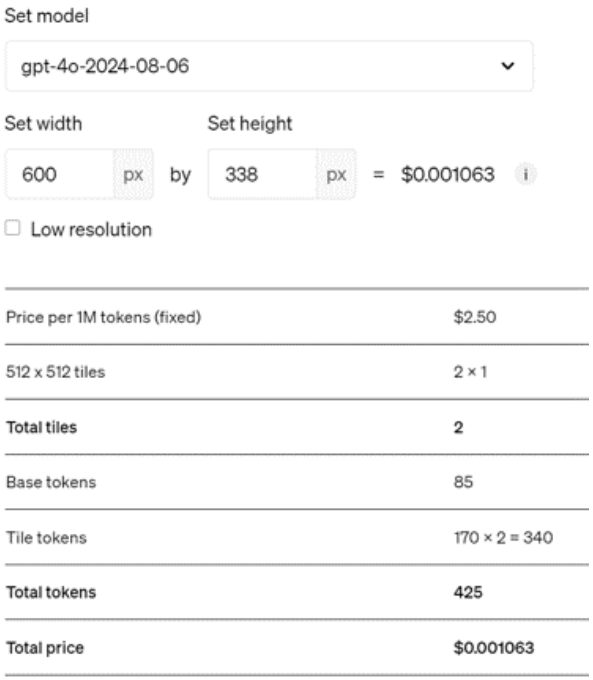

Ilustración 83: Consumo de tokens de gpt-4o

En la figura 83 se muestra el consumo de Tokens GPT-4o

6.3.3 Consumo de Tokens para el Procesamiento de Imágenes con GPT-4o-mini

Ilustración 84: Consumo de tokens de gpt-4o-mini

En la figura 84 se muestra el consumo de Tokens GPT-4o-mini

6.3.4 Resultado del Cálculo del Coste del Prompt de Entrada

Tras ejecutar un programa de python con el objetibo de contar los tokens del prompt que más adelante utilizaríamos para las pruebas, obtuvimos el siguiente resultado:

"The total input token consumption of the prompt will be: 327"

6.4 Primeras Pruebas (Modelo GPT-4o-mini)

6.4.1 Fotos Probadas

Primera Foto

Ilustración 85: Foto de pruebas 1

Foto con brotes (Ilustración 85).

Resultado de la prueba en la tabla 11:

Tabla 11: Respuesta a la foto de pruebas 1 con GPT-4o-mini

¿Ha Germinado?	¿Es lo Suficientemente Grande?	¿Tiene Alguna Enfermedad?	Enfermedad Detectada
Sí	No	No	Ninguna

Segunda Foto

Ilustración 86: Foto de pruebas 2

Foto sin brotes (Ilustración 86).

Resultado de la prueba en la tabla 12:

Tabla 12: Respuesta a la foto de pruebas 2 con GPT-4o-mini

¿Ha Germinado?	¿Es lo Suficientemente Grande?	¿Tiene Alguna Enfermedad?	Enfermedad Detectada
No	No	No	Ninguna

Tercera Foto

Ilustración 87: Foto de pruebas 3

Foto con Tizón Tardío (Ilustración 87).

Resultado de la prueba en la tabla 13:

Tabla 13: Respuesta a la foto de pruebas 3 con GPT-4o-mini

¿Ha Germinado?	¿Es lo Suficientemente Grande?	¿Tiene Alguna Enfermedad?	Enfermedad Detectada
Sí	No	Sí	Tizón

Cuarta Foto

Ilustración 88: Foto de pruebas 4

Foto con Mancha Bacteriana (Ilustración 88).

Resultado de la prueba en la tabla 14:

Tabla 14: Respuesta a la foto de pruebas 4 con GPT-4o-mini

¿Ha Germinado?	¿Es lo Suficientemente Grande?	¿Tiene Alguna Enfermedad?	Enfermedad Detectada
Sí	Sí	Sí	Mancha

6.4.2 Predicciones de Tokens

- El consumo de tokens para la cuarta foto utilizando GPT-4o-mini se predijo en aproximadamente 14.167 tokens para la imagen y 327 para el prompt.
- Se encontraron pequeñas discrepancias entre el consumo predicho y el real debido al script de predicción, ya que el payload fue copiado y pegado para ser analizado, pero el prompt real no incluye algunos símbolos usados en el formato JSON para enviar la solicitud.

6.5 Pruebas con el Modelo GPT-4o

6.5.1 Fotos Probadas

Tercera Foto (Modelo GPT-4o)

Foto con Tizón Tardío.

Ilustración 89: Foto de pruebas 3

Foto con Tizón Tardío (Ilustración 89).

Resultado de la prueba en la tabla 15:

Tabla 15: Respuesta a la foto de pruebas 3

¿Ha Germinado?	¿Es lo Suficientemente Grande?	¿Tiene Alguna Enfermedad?	Enfermedad Detectada
Sí	Sí	Sí	Pudrición

Cuarta Foto (Modelo GPT-4o)

Foto con Mancha Bacteriana.

Ilustración 90: Foto de pruebas 4

Foto con Mancha Bacteriana (Ilustración 90).

Resultado de la prueba en la tabla 16:

Tabla 16: Respuesta a la foto de pruebas 4

¿Ha Germinado?	¿Es lo Suficientemente Grande?	¿Tiene Alguna Enfermedad?	Enfermedad Detectada
Sí	Sí	Sí	Pudrición

Quinta Foto (Modelo GPT-4o)

Foto con pequeños brotes de tomate (Ilustración 91).

Ilustración 91: Foto de pruebas 5

Resultado de la prueba en la tabla 17:

Tabla 17: Respuesta a la foto de pruebas 5

¿Ha Germinado?	¿Es lo Suficientemente Grande?	¿Tiene Alguna Enfermedad?	Enfermedad Detectada
Sí	No	No	Ninguna

Sexta Foto (Modelo GPT-4o)

Foto con brote de tomate más grande (Ilustración 92).

Ilustración 92: Foto de pruebas 6

Resultado de la prueba en la tabla 18:

Tabla 18: Respuesta a la foto de pruebas 6

¿Ha Germinado?	¿Es lo Suficientemente Grande?	¿Tiene Alguna Enfermedad?	Enfermedad Detectada
Sí	No	No	Ninguna

Séptima Foto (Modelo GPT-4o)

Foto con una planta con crecimiento medio (Ilustración 93).

Ilustración 93: Foto de pruebas 7

Resultado de la prueba en la tabla 19:

Tabla 19: Respuesta a la foto de pruebas 7

¿Ha Germinado?	¿Es lo Suficientemente Grande?	¿Tiene Alguna Enfermedad?	Enfermedad Detectada
Sí	Sí	No	Ninguna

6.5.2 Análisis de Costes

- **Predicciones de Tokens**:
    - El consumo de tokens para la cuarta foto utilizando GPT-4o fue predicho en aproximadamente 581 tokens para la imagen y 327 para el prompt.
    - Se encontraron pequeñas discrepancias entre el consumo predicho y el real debido al script de predicción, ya que el payload fue copiado y pegado para ser analizado, pero el prompt real no incluye algunos símbolos usados en el formato JSON para enviar la solicitud.
- **Conclusiones**:
    - Para las primeras pruebas, se utilizó el modelo GPT-4o-mini debido a su menor coste por token en euros.
    - Sin embargo, el modelo GPT-4o-mini tiene un consumo mucho más alto de tokens para el análisis de imágenes, lo que incrementa el coste económico.
    - Comparando el consumo de tokens y el coste por token de cada modelo, el precio por token del procesamiento de imágenes del GPT-4o es menor que el de la versión mini. No obstante, dado que el prompt tiene el mismo número de tokens para ambos modelos y el precio del GPT-4o es más alto, el coste final es muy similar.
    - El coste promedio por solicitud fue de aproximadamente **0.0023 euros** para GPT-4o-mini y **0.0039 euros** para GPT-4o.

Dado que el coste de los dos modelos es muy similar, se utilizará el modelo GPT-4o, ya que es más potente y dará siempre mejores resultados que su versión mini.

6.5.3 Análisis de Calidad de Respuesta

- La respuesta se entrega en una estructura JSON como se solicitó, proporcionando 3 respuestas de tipo verdadero/falso y una respuesta abierta para la enfermedad que pueda tener la planta.

- Las respuestas del modelo GPT-4o fueron mejores que las del modelo mini, ya que puede diferenciar si la planta ha crecido lo suficiente no solo por el color del tomate, sino por el tamaño de la planta.
- La capacidad para identificar si la planta padece una enfermedad es buena, pero no es muy precisa al indicar de qué enfermedad se trata. Mientras que el modelo mini no ofrece una respuesta muy específica, el modelo 4o proporciona una respuesta más detallada, aunque no altamente precisa.

6.5.4 Mejoras a Realizar tras el Primer Análisis

- Con un prompt más sofisticado, se puede instruir a GPT para que elija únicamente entre tres o cuatro enfermedades más reconocibles, limitando así las respuestas inexactas (alucinaciones).

6.6 Reconocimiento de Enfermedades

Después de mejorar la ingeniería del prompt, se volvieron a probar las capacidades de reconocimiento de enfermedades. Para esta prueba se utilizaron las fotos 8-11 para evaluar la capacidad de distinguir entre diferentes enfermedades. También se volvieron a probar las fotos 3 (Tizón Tardío) y 4 (Mancha Bacteriana), ya que en pruebas anteriores se obtuvieron respuestas inexactas. Finalmente, se probó la foto 5, correspondiente a una planta sana, después de analizar consecutivamente fotos de plantas enfermas, para verificar si se obtenía una buena respuesta y que las respuestas previas no afectaban a las siguientes.

6.6.1 Fotos Probadas

Octava Foto

Foto con Tizón Tardío (Ilustración 94).

Ilustración 94: Foto de pruebas 8

Resultado de la prueba en la tabla 20:

Tabla 20: Respuesta a la foto de pruebas 8

¿Ha Germinado?	¿Es lo Suficientemente Grande?	¿Tiene Alguna Enfermedad?	Enfermedad Detectada
Sí	Sí	Sí	Tizón Tardío

Novena Foto

Foto con Mildiu Polvoriento (Ilustración 95).

Ilustración 95: Foto de pruebas 9

Resultado de la prueba en la tabla 21:

Tabla 21: Respuesta a la foto de pruebas 9

¿Ha Germinado?	¿Es lo Suficientemente Grande?	¿Tiene Alguna Enfermedad?	Enfermedad Detectada
Sí	Sí	Sí	Mildiu Polvoriento

Décima Foto

Foto con Mancha Bacteriana (Ilustración 96).

Ilustración 96: Foto de pruebas 10

Resultado de la prueba en la tabla 22:

Tabla 22: Respuesta a la foto de pruebas 10

¿Ha Germinado?	¿Es lo Suficientemente Grande?	¿Tiene Alguna Enfermedad?	Enfermedad Detectada
Sí	Sí	Sí	Mancha Bacteriana

Undécima Foto

Foto con Fusarium (Ilustración 97).

Ilustración 97: Foto de pruebas 11

Resultado de la prueba en la tabla 23:

Tabla 23: Respuesta a la foto de pruebas 11

¿Ha Germinado?	¿Es lo Suficientemente Grande?	¿Tiene Alguna Enfermedad?	Enfermedad Detectada
Sí	Sí	Sí	Fusarium

Tercera Foto

Foto con Tizón Tardío (Ilustración 98).

Ilustración 98: Foto de pruebas 3

Resultado de la prueba en la tabla 24:

Tabla 24: Respuesta a la foto de pruebas 3 tras la mejora de prompt

¿Ha Germinado?	¿Es lo Suficientemente Grande?	¿Tiene Alguna Enfermedad?	Enfermedad Detectada
Sí	Sí	Sí	Tizón Tardío

Cuarta Foto

Foto con Mancha Bacteriana (Ilustración 99).

Ilustración 99: Foto de pruebas 4

Resultado de la prueba en la tabla 25:

Tabla 25: Respuesta a la foto de pruebas 4 tras la mejora del prompt

¿Ha Germinado?	¿Es lo Suficientemente Grande?	¿Tiene Alguna Enfermedad?	Enfermedad Detectada
Sí	Sí	Sí	Mancha Bacteriana

Quinta Foto

Foto de una planta sana (Ilustración 100).

Ilustración 100: Foto de pruebas 5

Resultado de la prueba en la tabla 26:

Tabla 26: Respuesta a la foto de pruebas 5 tras la mejora del prompt

¿Ha Germinado?	¿Es lo Suficientemente Grande?	¿Tiene Alguna Enfermedad?	Enfermedad Detectada
Sí	No	No	Ninguna

6.6.2 Conclusión

Ahora contamos con una función de detección de enfermedades muy robusta. Después de las pruebas podemos asegurar que el sistema es capaz de diferenciar entre varias enfermedades sin generar respuestas inexactas para plantas sanas.

6.7 Pruebas de Casos Límite

6.7.1 Mejoras

Se modificó la ingeniería del prompt y se añadió un código de manejo de errores para obtener mensajes de error en casos donde no se detectará una planta o maceta.

Duodécima Foto

Foto con un vidrio empañado (Ilustración 101).

Ilustración 101: Foto de pruebas 12

Resultado de la prueba: Error: no se detectó ninguna planta o maceta.

Decimotercera Foto

Foto de jugadores del Bayern FC en el Oktoberfest (Ilustración 102).

Ilustración 102: Foto de pruebas 13

Resultado de la prueba: Error: no se detectó ninguna planta o maceta.

Decimocuarta Foto

Foto de una planta con un ambiente brumoso y con algunos garabatos (Ilustración 103).

Ilustración 103: Foto de pruebas 14

Resultado de la prueba: Error: no se detectó ninguna planta o maceta.

Decimoquinta Foto

Foto de una planta con un ambiente brumoso sin garabatos (Ilustración 104).

Ilustración 104: Foto de pruebas 15

Resultado de la prueba en la tabla 27:

Tabla 27: Respuesta a la foto de pruebas 15

¿Ha Germinado?	¿Es lo Suficientemente Grande?	¿Tiene Alguna Enfermedad?	Enfermedad Detectada
Sí	No	No	Ninguna

6.7.2 Conclusión

Para esta prueba se utilizaron las cuatro fotos de casos límite, así como las fotos 11, 6 y 2, para asegurarse de que no se generaran respuestas inexactas en las otras capacidades de reconocimiento al añadir esta mejora. Todas las respuestas fueron las esperadas.

6.8 Ingeniería de Prompt en la Llamada a la API de GPT

Este apartado analiza las prácticas de ingeniería de prompt implementadas en la solicitud a la API de OpenAI (GPT-4o). Se estudian las estrategias utilizadas para mejorar la precisión y relevancia de las respuestas del modelo, garantizando que el sistema interprete correctamente las imágenes de plantas en un contexto de cuidado de germinación.

6.8.1 Diseño de la Ingeniería de Prompt

Uso del Rol "System" para Configuración del Comportamiento

El primer mensaje en la estructura de messages define el comportamiento del asistente de IA mediante el rol "system". Este enfoque es clave en la ingeniería de prompt porque: - **Establece un contexto claro**: Define el modelo como un experto en análisis de plantas germinadas. - **Reduce la ambigüedad**: Indica explícitamente que la imagen debe contener una planta o maceta; si no, devuelve un mensaje de error. - **Define criterios de análisis**: Se especifica que el modelo debe evaluar la germinación, el crecimiento y posibles enfermedades.

```
{
   "role": "system",
   "content": [
      {
         "type": "text",
         "text": "You are an expert plant care-taker system that analyzes the stat
e of sprouting plants. You should ensure that the image contains a plant or po
t. If not, return the message 'no plant or pot has been detected'. If there is a pl
ant, analyze whether it has germinated, if it is grown enough, or if it has a dise
ase. Return the information in json_format."
      }
   ]
}
```

- Traducción del prompt al español: Eres un sistema experto en el cuidado de plantas que analiza el estado de las plantas en germinación. Debes asegurarte de que la imagen contenga una planta o una maceta. Si no es así, devuelve el mensaje “no se ha detectado ninguna planta o maceta”. Si hay una planta, analiza si ha germinado, si ha crecido lo suficiente o si tiene alguna enfermedad. Devuelve la información en formato JSON.

Rol “User” para Solicitar un Análisis Estructurado

El segundo mensaje, bajo el rol "user", contiene la solicitud del usuario. Aquí se aplican buenas prácticas clave:

Especificidad en la Solicitud

El prompt solicita una respuesta estructurada en un formato JSON claro, evitando interpretaciones erróneas del modelo.

```
{
   "role": "user",
   "content": [
      {
         "type": "text",
         "text": """Extract the state of the plant from an image and provide a stru
ctured response. The information should include: has the plant germinated? b
oolean true/false /// is the plant big enough to be taken off the germination
module? boolean true/false /// does the plant have a disease? boolean true/
false /// in case of a disease, specify one of the common diseases (late blight
, powdery mildew, bacterial spot, fusarium wilt), otherwise specify 'unknown' if
unidentifiable, or 'none' if no disease is detected. If the image does not contai
n a plant or pot, respond with the error message: 'no plant or pot has been det
ected'. The format will be:
```

```
                ```json
                [{
                "germinated": "<true/false>",
                "grown": "<true/false>",
                "disease": "<true/false>",
                "whatdisease": "name of the disease"
                }]"""
    }
  ]
}
```

- Traducción del prompt al español: Extrae el estado de la planta a partir de una imagen y proporciona una respuesta estructurada. La información debe incluir: ¿Ha germinado la planta? (booleano true/false) /// ¿Es la planta lo suficientemente grande como para ser retirada del módulo de germinación? (booleano true/false) /// ¿La planta tiene alguna enfermedad? (booleano true/false) /// En caso de que tenga una enfermedad, especifica una de las enfermedades comunes (tizón tardío, oídio, mancha bacteriana, fusariosis); de lo contrario, especifica 'desconocido' si no es identificable, o 'ninguna' si no se detecta ninguna enfermedad. Si la imagen no contiene una planta o una maceta, responde con el mensaje de error: 'no se ha detectado ninguna planta o maceta'. El formato será:

```
[{
"germinado":                    "<verdadero/falso>",
"crecido":                      "<verdadero/falso>",
"germinado":                    "<verdadero/falso>",
"queenfermedad":    "nombre    de    la    enfermedad"
}]
```

Lista de Preguntas Cerradas

Se establecen preguntas binarias (true/false) sobre el estado de la planta, lo que:

- Minimiza respuestas vagas.
- Facilita la interpretación automatizada de los datos.

```
{
  "germinated": "<true/false>",
  "disease": "<true/false>",
  "grown": "<true/false>",
  "whatdisease": "name of the disease"
}
```
```

Manejo de Casos de Error desde el Prompt

Si la imagen no contiene una planta o maceta, se instruye al modelo a devolver el mensaje 'no plant or pot has been detected'. Esto mejora la robustez del sistema frente a entradas inválidas.

6.8.2 Análisis de la Incorporación de la Imagen

El prompt incluye una imagen en **formato base64**, estructurada de la siguiente manera:

```
{
   "type": "image_url",
   "image_url": {
      "url": "{image_base64}",
      "detail": "high"
   }
}
```

Buenas prácticas en la integración de imágenes

- **Uso del nivel de detalle "high"**: Esto maximiza la calidad de análisis de la IA.
- **Formato base64**: Permite la transmisión de imágenes sin necesidad de almacenamiento en servidores externos.

6.8.3 Parámetro de Control de Respuesta

Para evitar respuestas excesivamente largas y optimizar el uso de tokens, se ha limitado la respuesta a max_tokens: 150. Esto:

- Reduce el coste computacional.
- Mantiene respuestas concisas y eficientes.

6.8.4 Conclusión

La ingeniería de prompt aplicada en esta solicitud de API optimiza la precisión y estructura de las respuestas de GPT mediante:

1. **Definición clara del contexto** en el mensaje del sistema.
2. **Uso de preguntas específicas y formato JSON estructurado** en la solicitud del usuario.
3. **Gestión de imágenes con detalles específicos** para mejorar la interpretación.
4. **Manejo explícito de errores** para robustez operativa.

Este diseño garantiza un flujo de información confiable y útil para la automatización del análisis de germinación de plantas.

6.9 Conclusión

- Después de ajustar la ingeniería del prompt, la API responde con datos precisos.
- Las pruebas de casos límite arrojaron resultados muy positivos, ya que el sistema devuelve un mensaje de error cuando no hay una maceta o planta:
    - No genera respuestas inexactas para fotos no relacionadas.
    - Cuando la visibilidad es insuficiente, puede reconocer plantas, pero si no es posible, devuelve un mensaje de error.
    - Las mejoras en los casos límite no afectan la capacidad de reconocimiento para fotos regulares de plantas o macetas.
- Después de todas las pruebas y predicciones de costes, podemos asegurar que al usar el modelo GPT-4o el coste por cada 10 fotos sería menor a 4 centavos de euro. Además, si reducimos la calidad de las fotos, el precio podría disminuir hasta 9 veces, lo que permitiría regular los costes de las pruebas disminuyendo la calidad de las fotos y aumentándola para operaciones regulares si es necesario.
- Esta solución no solo es precisa, completa y fácil de implementar, sino que también es flexible y fácil de adaptar a nuevos entornos. Esto puede comprobarse en el apartado A3.4 del Anejo 4, donde se adapta esta funcionalidad para realizar el reconocimiento del estado de plantas de un robot invernadero de ITQ.

Aunque los resultados de la API de GPT muestran precisión y una alta calidad de respuestas, se ha llevado a cabo una investigación exhaustiva de otras opciones, como son Gemini, Llava, y el uso de Llava y Llama3.1 conjuntamente. El proceso de pruebas y los resultados se muestran en el Anejo 3.

7 Desarrollo del Asistente por Voz con un Modelo LLM

Dado que el proyecto a nivel interno de la empresa tenía como objetivo investigar nuevos métodos de IA para el control de robots, y, además, uno de nuestros objetivos era simplificar la interacción con el usuario, decidimos investigar la viabilidad de realizar un control por voz con lenguaje natural que permitiera al usuario hablar con el robot sin necesidad de memorizar y decir comandos específicos, sino hablando como hablaría con otra persona.

Para ello investigamos herramientas para realizar las funcionalidades de texto a habla y de habla a texto, cosa que fue sencilla, ya que hay varias opciones de código abierto.

Sin embargo, el mayor reto era encontrar la forma de realizar el procesamiento de lenguaje natural. Desde el principio nuestro objetivo fue utilizar un LLM, idealmente uno que pudiéramos ejecutar en local en los servidores de la compañía, para no depender de terceros. Uno de los mayores retos era encontrar la forma de darle suficiente contexto al LLM para poder dar respuestas precisas, ya que nuestro sistema tiene muchas tareas posibles y simplemente con un solo prompt era muy complicado obtener respuestas precisas y evitar las famosas 'alucinaciones' de los LLM. Al principio probamos a adaptar unos programas de prueba sencillos, que se usaron en la compañía para otro proyecto, basados en RAG (Retrieval-Augmented Generation). Tras adaptar los programas a nuestro caso de uso y probarlos, se comprobó que no eran útiles para controlar nuestro sistema mecatrónico.

Antes de investigar más sobre RAG para mejorar los programas, decidí probar la idea de utilizar un árbol de decisiones con varias llamadas al LLM consecutivas, que permitieran cambiar el contexto de forma dinámica. Las primeras pruebas de este método fueron muy positivas, lo cual nos hizo enfocarnos en esta solución y finalmente desarrollar el asistente por voz que se analiza en este capítulo.

El Asistente de Voz con Inteligencia Artificial permite el control interactivo de un sistema mecatrónico diseñado para el cuidado automatizado de plantas. Este asistente utiliza **LLM con Ollama** como modelo base **[20]**. Integra:

- **OpenAI Whisper** para la transcripción de voz a texto.
- **pyttsx3** para la conversión de texto a voz.
- **Llama3.1** como modelo LLM para el procesamiento del lenguaje natural y la obtención de comandos específicos para el sistema.

El asistente escucha comandos de voz, los procesa en tareas ejecutables y responde con los comandos correspondientes al sistema.

7.1 Características Principales

- **Conversión de Voz a Texto**: Transforma comandos hablados en texto utilizando Whisper.
- **Procesamiento de Lenguaje Natural**: Clasifica los comandos y proporciona respuestas adecuadas.
- **Conversión de Texto a Voz**: Comunica los resultados de manera audible utilizando pyttsx3.
- **Gestión de Tareas**: Soporta tareas clasificadas en:
    - **No relacionadas**: Consultas de conocimiento general.
    - **Robot**: Comandos para manipulación de plantas e interacción con módulos.
    - **Entradas/Salidas (IO)**: Ajustes de variables ambientales.
    - **Información**: Consultas sobre datos de sensores y estado del sistema.

7.2 Desglose del Código

7.2.1 Bibliotecas y Dependencias

```
import ollama
import whisper
import sounddevice as sd
import numpy as np
import tempfile
import os
from scipy.io.wavfile import write
import pyttsx3
import time
```

- **ollama**: Gestiona la comunicación con el LLM para clasificar comandos y generar respuestas.
- **whisper**: Transcribe el audio de entrada a texto.
- **sounddevice**: Graba audio desde un micrófono.
- **numpy & scipy.io.wavfile**: Procesa datos de audio.
- **pyttsx3**: Convierte respuestas en texto a formato de voz.

7.2.2 Inicialización

ollama_client = ollama.Client(host="Aquí debe incluirse la dirección del servido
r ejecutando Ollama")

- Conecta al servidor de Ollama LLM, que se ejecuta en local en un ordenador dentro de la compañía, para gestionar tareas conversacionales.

7.2.3 Capa 0: Clasificación de Comandos

Propósito

Clasifica los comandos del usuario en una de las cuatro categorías: - no relacionadas: Consultas generales ajenas al sistema. - robot: Comandos para el control del robot (por ejemplo, mover, sembrar, fotografiar). - io: Comandos para modificar configuraciones ambientales (por ejemplo, temperatura, humedad). - información: Consultas sobre el estado del sistema o datos de sensores.

Función Principal

La llamada a la API incluye una lista de ejemplos de posibles comandos de usuario y la respuesta correcta que el asistente debería proporcionar. Esto mejora la calidad de las respuestas al proporcionar al LLM un contexto previo. La estructura de la llamada a la API es la siguiente:

```
def layer0_ollama_call(prompt: str):
    response = ollama_client.chat(
        model="llama3.1:8b",
        messages=[
            # Ejemplos de comandos categorizados
            {"role": "user", "content": "Move the pot 1 to the table 2"},
            {"role": "assistant", "content": "robot"},
            ...
            {"role": "user", "content": prompt},
        ],
        options={"seed": 101, "temperature": 0}
    )
    return response["message"]["content"]
```

- Recibe un prompt y devuelve una respuesta de una sola palabra indicando la categoría.
- El prompt que recibe el LLM tiene la siguiente estructura: python full_prompt = layer0_who_are_you_prompt + layer0_task_prompt + prompt Donde las diferentes partes son:
    - Los posibles tipos de respuesta:

```
layer0_responses = ["unrelated", "robot", "io", "info"]
```

- Una descripción del rol del asistente:

```
layer0_who_are_you_prompt = (
    "You are an automated assistant working in a mechatronic system "
    "for automated plant care. You will control the different parts "
    "of the system, controlling a robot, controlling the IO of the "
    "system or providing information."
)
```

- El prompt específico de la tarea:

```
layer0_task_prompt = f"""
An operator is going to ask you for a task and you should answer which
kind of request from this list {layer0_responses} best fits the operator's
command. Answer with ONE ONLY WORD FROM THE LIST. DO NOT repl
y with
any word other than those in the list. Consider that the command come
s
from a speech-to-text transcript, so some words might not make sense.
You should figure out by the context what the operator really meant in
such cases.
. unrelated: the user might have asked something about history, the
    weather, sports, science or any other unrelated thing.
. robot: the user might have asked the assistant to move one pot or
    plant, plant a seed inside a pot, or take a photo of one of the plants.
. io: the user might have asked the assistant to turn on or off one of
    the actuators or modify target values for light, temperature, humidity,
    CO2, or irrigation.
. info: the user might have asked the assistant about sensor readings
    (light, temperature, humidity, CO2) or the state of a plant (e.g.,
    whether it has germinated).
Please, ANSWER WITH ONLY ONE WORD OF THE LIST.
"""
```

- El comando de voz transcrito se pasa como un parámetro: prompt.

7.2.4 Capa 1: Gestión Específica de Tareas

Cada tipo de comando tiene sus propios prompts contextuales: layer1_responses_<tipo>, layer1_who_are_you_prompt_<tipo>, layer1_task_prompt_<tipo>. Esta estructura en capas permite proporcionar al LLM un contexto más preciso para cada caso de comando de voz. Se utiliza el mismo comando de voz, pero se ajusta el contexto según el tipo de solicitud

que recibe el asistente. Esto minimiza respuestas imprecisas o “alucinaciones” del LLM.

Comandos No Relacionados

Responde preguntas de conocimiento general con respuestas concisas.

```
def layer1_ollama_call_unrelated(prompt: str):
    response = ollama_client.chat(
        model="llama3.1:8b",
        messages=[
            {"role": "user", "content": "Who won the 2010 football world cup"},
            {"role": "assistant", "content": "Spain. ..."},
            ...
            {"role": "user", "content": prompt},
        ],
        options={"seed": 101, "temperature": 0.2}
    )
    return response["message"]["content"]
```

Comandos para el Robot

Identifica tareas específicas relacionadas con el robot, tales como:

- Mover macetas o plantas (takein, takeout).
- Plantar semillas (seed).
- Tomar fotografías (photo).

```
def layer1_ollama_call_robot(prompt: str):
    response = ollama_client.chat(
        model="llama3.1:8b",
        messages=[
            {"role": "user", "content": "Take the plant two out of the module"},
            {"role": "assistant", "content": "takeout, pot2"},
            ...
            {"role": "user", "content": prompt},
        ],
        options={"seed": 101, "temperature": 0}
    )
    return response["message"]["content"]
```

Comandos IO (Entradas y Salidas)

Procesa comandos para ajustar variables ambientales: - light (luz), temperature (temperatura), co2, humidity (humedad), irrigation (irrigación).

```
def layer1_ollama_call_io(prompt: str):
    response = ollama_client.chat(
```

```
        model="llama3.1:8b",
        messages=[
            {"role": "user", "content": "Set the Co2 levels to 800 ppm"},
            {"role": "assistant", "content": "co2, 800"},
            ...
            {"role": "user", "content": prompt},
        ],
        options={"seed": 101, "temperature": 0}
    )
    return response["message"]["content"]
```

Comandos de Información

Distingue entre consultas relacionadas con el robot y datos de sensores.

```
def layer1_ollama_call_info(prompt: str):
    response = ollama_client.chat(
        model="llama3.1:8b",
        messages=[
            {"role": "user", "content": "How are the co2 levels right now?"},
            {"role": "assistant", "content": "sensor"},
            ...
            {"role": "user", "content": prompt},
        ],
        options={"seed": 101, "temperature": 0}
    )
    return response["message"]["content"]
```

7.2.5 Árbol de Decisiones

Resumen

Gestiona la toma de decisiones en capas:

1. Clasifica el comando (Capa 0).
2. Procesa la tarea dentro de la categoría identificada y con un contexto específico (Capa 1).

En la ilustración 105 se puede ver el diagrama del árbol de decisiones.

Diagrama

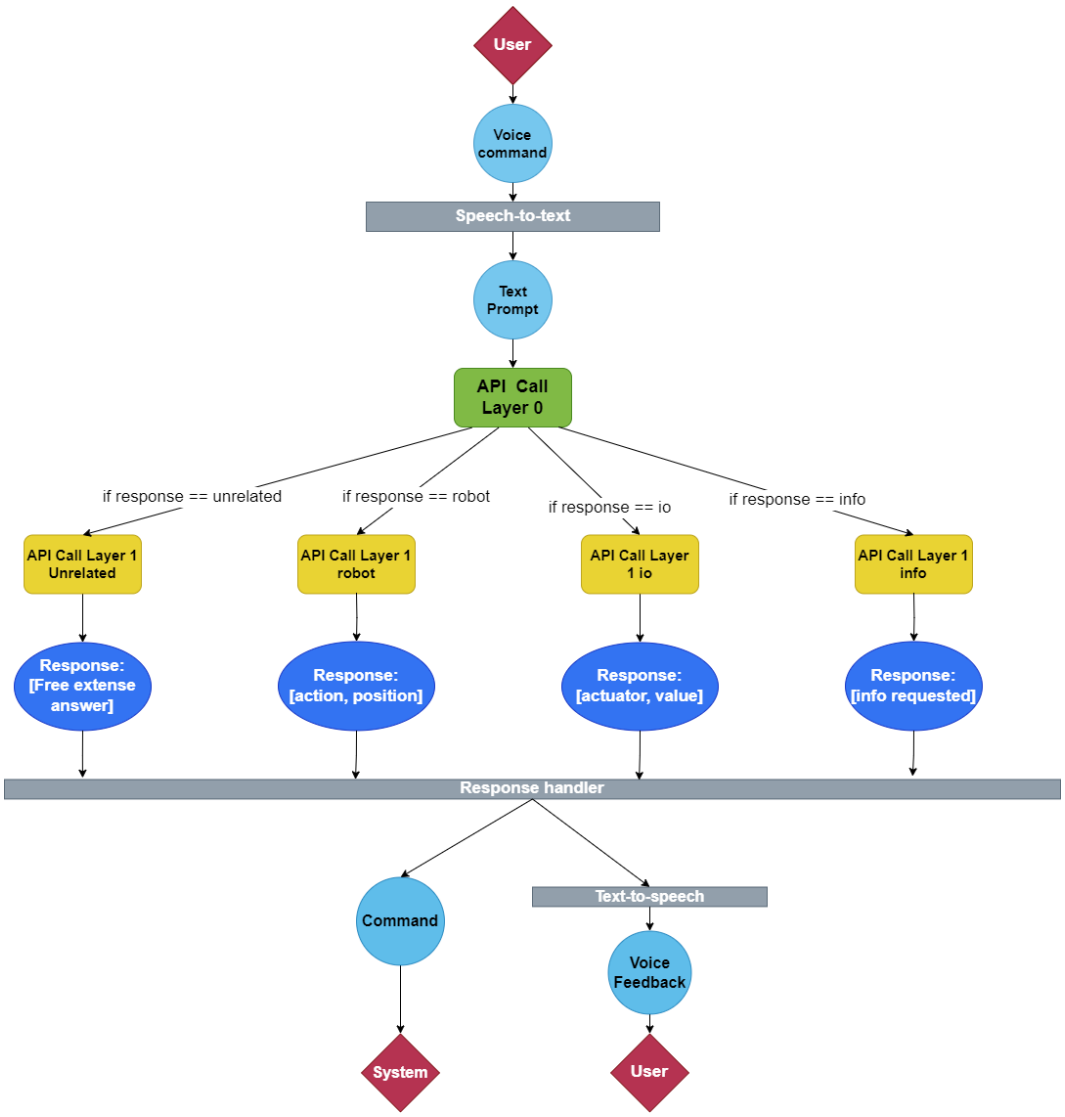

Ilustración 105: Diagrama del árbol de decisiones del sistema con lenguaje natural

Desglose del Código

```
# Función principal del árbol de decisiones para gestionar el prompt y llamar a las capas del LLM
def decision_tree(prompt: str):
    # Construir el prompt completo para la Capa 0
    full_prompt = layer0_who_are_you_prompt + layer0_task_prompt + prompt

    # Iniciar el temporizador para la Capa 0
    start_time = time.time()
    response = layer0_ollama_call(full_prompt)
    end_time = time.time()  # Finalizar el temporizador

    # Calcular y formatear el tiempo transcurrido para la Capa 0
```

```
    elapsed_time_layer0 = end_time - start_time
    formatted_time_layer0 = format_time(elapsed_time_layer0)
    print(f"La Capa 0 del LLM tomó {formatted_time_layer0} en responder.")
    print("Esta solicitud pertenece al siguiente tipo:", response)
    text_to_speech(f"Esta solicitud pertenece al siguiente tipo: {response}")

    # Gestionar llamadas adicionales según el tipo de respuesta
    if response == 'unrelated':
        # Construir el prompt completo para la Capa 1 (no relacionada)
        full_prompt = layer1_who_are_you_prompt_unrelated + layer1_task_pro
mpt_unrelated + prompt

        # Iniciar el temporizador para la Capa 1
        start_time = time.time()
        response1 = layer1_ollama_call_unrelated(full_prompt)
        end_time = time.time()  # Finalizar el temporizador

        # Calcular y formatear el tiempo transcurrido para la Capa 1
        elapsed_time_layer1 = end_time - start_time
        formatted_time_layer1 = format_time(elapsed_time_layer1)
        print(f"La Capa 1 del LLM (no relacionada) tomó {formatted_time_layer1}
en responder.")

        print("Respuesta:", response1)
        text_to_speech(f"{response1}")

    elif response == 'robot':
        # Construir el prompt completo para la Capa 1 (robot)
        full_prompt = layer1_who_are_you_prompt_robot + layer1_task_prompt_
robot + prompt

        # Iniciar el temporizador para la Capa 1
        start_time = time.time()
        response1 = layer1_ollama_call_robot(full_prompt)
        end_time = time.time()  # Finalizar el temporizador

        # Calcular y formatear el tiempo transcurrido para la Capa 1
        elapsed_time_layer1 = end_time - start_time
        formatted_time_layer1 = format_time(elapsed_time_layer1)
        print(f"La Capa 1 del LLM (robot) tomó {formatted_time_layer1} en respo
nder.")

        print("Respuesta:", response1)
```

```
        text_to_speech(f"{response1}")
    ...
```

7.2.6 Grabación y Transcripción de Audio

Grabar y Guardar Audio

```
# Grabar audio desde el micrófono
def record_audio(duration=7, fs=44100):
    print("Grabando... Por favor, diga su comando.")
    audio = sd.rec(int(duration * fs), samplerate=fs, channels=1, dtype='float3
2')
    sd.wait()
    print("Grabación completada.")
    return np.squeeze(audio)
```

Guardar el Audio

```
# Guardar el audio en un archivo temporal
def save_audio_to_file(audio, fs):
    temp_file = tempfile.NamedTemporaryFile(delete=False, suffix=".wav")
    write(temp_file.name, fs, audio)
    return temp_file.name
```

Transcribir el Audio

```
def transcribe_audio(file_path):
    model = whisper.load_model("base")
    result = model.transcribe(file_path, language="en")
    return result.get("text", "")
```

Obtener el Texto del Comando de Voz

```
# Función principal para gestionar la conversión de voz a texto
def get_speech_to_text():
    duration = 7  # Grabar durante 7 segundos
    fs = 44100
    audio = record_audio(duration, fs)
    file_path = save_audio_to_file(audio, fs)
    text = transcribe_audio(file_path)
    os.remove(file_path)  # Eliminar el archivo temporal
    return text
```

7.2.7 Conversión de Texto a Voz

Convierte las respuestas del LLM en audio.

```
def text_to_speech(text):
    engine = pyttsx3.init()
```

```
    engine.say(text)
    engine.runAndWait()
```

7.2.8 Función Principal

Integra todos los componentes del sistema:

```
def main():
    print("Iniciando el Sistema de Comandos de Voz...")

    # Paso 1: Obtener voz a texto
    print("Paso 1: Grabando y Transcribiendo Voz...")
    transcribed_text = get_speech_to_text()
    print("Texto Transcrito:", transcribed_text)

    # Paso 2: Procesar con Ollama
    print("Paso 2: Analizando el comando con LLM...")

    decision_tree(transcribed_text)
```

7.3 Análisis de Ingeniería del Prompt en el Control por Voz con Ollama

Este apartado analiza la ingeniería del prompt en el sistema de control por voz de un sistema mecatrónico utilizando **Ollama**. Se enfoca en los prompts implementados en **Layer 0** y **Layer 1_robot**.

7.3.1 Layer 0: Clasificación de Solicitudes

Listas de posibles respuestas cerradas

En Layer 0, se define una lista de respuestas cerradas para clasificar las solicitudes del operador:

```
layer0_responses = ["unrelated", "robot", "io", "info"]
```

Cada categoría está bien definida en el prompt:

- **unrelated**: Peticiones no relacionadas con el sistema.
- **robot**: Comandos para mover macetas, plantar semillas o tomar fotos.
- **io**: Acciones sobre actuadores o ajustes de variables ambientales.
- **info**: Solicitudes sobre sensores, estado de plantas o estado del robot.

El prompt exige responder con una sola palabra de la lista, lo que reduce la ambigüedad y mejora el determinismo en la clasificación.

Who are you prompt

```
layer0_who_are_you_prompt = """
You are an automated assistant working in a mechatronic system for automat
ed plant care. You will control the different parts of the system, controlling a ro
bot, controlling the IO of the system or providing information.
"""
```

Este prompt establece claramente el rol del modelo como asistente de un sistema de cuidado automatizado de plantas. Es específico en cuanto a las funciones que puede realizar, lo que ayuda al modelo a centrarse en su dominio.

Prompt asignado por el usuario

```
layer0_task_prompt = f"""
A operator is going to ask you for a task and you should answer which kind of r
equest from this list {layer0_responses} best fits the operator's command. Ans
wer with ONE ONLY WORD FROM THE LIST. DO NOT reply with any word other t
han those in the list. Consider that the command comes from a speech-to-text
transcript, so is possible that sometimes somes words might not make sense,
you should figure out by the context what the operator really meant in this cas
es.
- unrelated: the user migh have asked something about history, the weather, s
ports, science or any other unrelated thing.
- robot: the user might have asked the assistant to move one pot or plant, plan
t a seed inside a pot or take a photo of one of the plants. All these request are
robot requests but any request related with the robot status is a info request.
- io: the user might have asked the assistant to turn on or off one of the followi
ng actuators: lights, heating cable, humidifier, fan or electrovalve (the electrov
alve is used for irrigation and watering the plants). The user might have asked
the assistant to modify one of the following target values: light, temperature, h
umidity, Co2 or irrigation respectively.
- info: the user might have asked the assitant about the reading of the sensors
, that could be: light, temperature, humidity, Co2. Or about the state of some o
f the plants, as for example, tell me if the plant one has germinated. Or for exa
mple the user migh have asked you about the robot state, For example: 'what i
s going on with the robot?' or "tell me the robot status."
Please, ANSWER WITH ONLY ONE WORD OF THE LIST"""
```

Este prompt:

- Enfatiza que solo debe responder con una palabra de la lista.
- Explica que los comandos pueden provenir de transcripciones imperfectas.

- Brinda ejemplos de cada categoría, lo que refuerza la interpretación correcta del contexto.

Historial de conversación y contexto

Al modelo se le da un mayor contexto con ejemplos previos para mejorar su clasificación:

```
messages=[
   {"role": "user", "content": "Move the pot 1 to the table 2"},
   {"role": "assistant", "content": "robot"},
   {"role": "user", "content": "Water the plants and set the temperature to 30"},
   {"role": "assistant", "content": "io"},
   ... Se añaden hasta alrededor de diez ejemplos...
]
```

Proporcionar ejemplos en la conversación ayuda a que el modelo generalice mejor.

7.3.2 Layer 1: Acciones del Robot

Listas de posibles respuestas cerradas

```
layer1_responses_robot = ["takein","takeout", "seed", "photo"]
```

Cada acción del robot tiene una categoría bien definida:

- **takein**: Mover una maceta al módulo.
- **takeout**: Sacar una maceta del módulo.
- **seed**: Plantar una semilla.
- **photo**: Tomar una foto de una planta.

El prompt sigue la misma estrategia de respuesta cerrada, reduciendo ambigüedad y asegurando un control estructurado del robot.

Who are you prompt

```
layer1_who_are_you_prompt_robot = f"""
You are an automated assistant working in a mechatronic system for automat
ed plant care. You will receive a transcripted voice command from the user an
d you will have to decide which kind of robot action is needed from this list {lay
er1_responses_robot}.
"""
```

Similar al prompt de Layer 0, establece el dominio del asistente, asegurando que solo se enfoque en las tareas del robot.

Prompt asignado por el usuario

```
layer1_task_prompt_robot = f"""
A operator is going give you a voice command (consider that the voice comma
nd is transcripted with a software that is not perfect and may fail, so if some w
ords are not coherent, you should figure out what the user meant by the conte
xt). The possible request are stablished in this list: {layer1_responses_robot}.
Answer with ONE WORD FROM THE LIST AND ONE VALUE SEPARATED BY A ",".
DO NOT reply with any word other than those in the list and the value. Here is
a further explanation of each kind of request:
- takein: you should reply with 'takein, <pot requested>' if the user tells you to
move a pot or a plant into the module. For example, "take the first plant into th
e module" or "move the pot three in the module"
- takeout: you should reply with 'takeout, <pot requested>' if the user tells you
to move a pot or a plant out of the module. For example, "take the plant 2 out
of the module" or "move the pot 3 outside"
- seed: you should reply with 'seed, <pot requested>' if the user tells you to pla
nt a seed into a pot. For example, "seed the first pot with a tomato" or "plant a
lettuce in the second pot".
- photo: you should reply with 'photo, <pot requested>' if the user request you
to take a photo of one of the plants. For example, "take a photo of the third pla
nt".
Consider that the <pot requested> field should always be filled with the word '
pot' and a number. For example: 'pot1' or 'pot3.
"""
```

Este prompt:

- Refuerza que el modelo debe interpretar comandos con errores de transcripción.
- Exige una respuesta estructurada en formato acción, potX.
- Proporciona ejemplos claros para cada acción.

Historial de conversación y contexto

Al modelo también se le aporta contexto con ejemplos previos:

```
messages=[
   {"role": "user", "content": "Take the plant two out of the module"},
   {"role": "assistant", "content": "takeout, pot2"},
   {"role": "user", "content": "Take a photo of the third plant"},
   {"role": "assistant", "content": "photo, pot3"},
   ... Se añaden hasta alrededor de diez ejemplos...
]
```

Estos ejemplos refuerzan la interpretación correcta del modelo y evitan errores en la clasificación de comandos.

7.3.3 Buenas Prácticas de Ingeniería del Prompt

3. **Uso de listas cerradas**: Reduce ambigüedad y mejora consistencia.
4. **Definición clara del rol del asistente**: Asegura que el modelo permanezca en su dominio de aplicación.
5. **Instrucciones específicas y concisas**: Uso de restricciones claras ("ONE ONLY WORD").
6. **Ejemplos previos en el contexto**: Mejora la capacidad del modelo para generalizar comandos correctamente.
7. **Explicación de errores de transcripción**: Permite al modelo corregir imprecisiones en los comandos de voz.
8. **Uso de estructura fija en las respuestas**: Evita respuestas libres y permite una fácil interpretación por parte del sistema.

7.3.4 Conclusión

La ingeniería del prompt en este sistema está bien diseñada para garantizar una interpretación precisa y estructurada de los comandos de voz. El uso de listas cerradas, ejemplos previos y restricciones claras minimiza errores y mejora la fiabilidad del control del sistema mecatrónico. Siguiendo estas buenas prácticas, el modelo se mantiene dentro de su dominio y proporciona respuestas predecibles y útiles.

7.4 Pruebas del Sistema de Control por Voz

Este apartado describe las pruebas realizadas en el sistema de control por voz, documentando las respuestas de cada layer y los tiempos de procesamiento registrados.

7.4.1 Prueba: Comando No Relacionado

Prueba 1: Pregunta General

Comando Transcrito:

Tell me why the sky is blue.

Respuesta

En la tabla 28 se muestra la respuesta de cada capa del árbol de decisiones al comando anterior:

Tabla 28: Respuesta de cada capa del árbol de decisiones para un comando no relacionado

Capa (Layer)	Tiempo de Respuesta	Respuesta
Layer 0 (LLM)	00:00:241	**Tipo de solicitud:** *unrelated*
Layer 1 (LLM - unrelated)	00:00:972	*El cielo es azul debido a la dispersión de Rayleigh...*

7.4.2 Pruebas: Robot

Prueba 2: Extracción de una Planta del Módulo

Comando Transcrito:

Take the second plant out of the module.

Respuesta

En la tabla 29 se muestra la respuesta de cada capa del árbol de decisiones al comando anterior:

Tabla 29: Respuesta de cada capa del árbol de decisiones para un comando take out del robot

Capa (Layer)	Tiempo de Respuesta	Respuesta
Layer 0 (LLM)	00:00:214	**Tipo de solicitud:** *robot*
Layer 1 (LLM - robot)	00:00:175	**Comando:** takeout, **Objeto:** pot2

Prueba 3: Tomar una Foto

Comando Transcrito:

Take a photo of the third pot.

Respuesta

En la tabla 30 se muestra la respuesta de cada capa del árbol de decisiones al comando anterior:

Tabla 30: Respuesta de cada capa del arbol de decisones para un comando photo del robot

Capa (Layer)	Tiempo de Respuesta	Respuesta
Layer 0 (LLM)	00:01:021	**Tipo de solicitud:** *robot*
Layer 1 (LLM - robot)	00:00:142	**Comando:** photo, **Objeto:** pot3

Prueba 4: Introducción de una Planta en el Módulo

Comando Transcrito:

Take the second plant into the module.

Respuesta

En la tabla 31 se muestra la respuesta de cada capa del árbol de decisiones al comando anterior:

Tabla 31: Respuesta de cada capa del árbol de decisiones para un comando take in del robot

Capa (Layer)	Tiempo de Respuesta	Respuesta
Layer 0 (LLM)	00:00:364	**Tipo de solicitud:** *robot*
Layer 1 (LLM - robot)	00:00:112	**Comando:** takein, **Objeto:** pot2

Prueba 5: Insertar una Semilla en una Maceta

Comando Transcrito:

Put a seed into the fourth pot.

Respuesta

En la tabla 32 se muestra la respuesta de cada capa del árbol de decisiones al comando anterior:

Tabla 32: Respuesta de cada capa del árbol de decisiones para un comando seed del robot

Capa (Layer)	Tiempo de Respuesta	Respuesta
Layer 0 (LLM)	00:00:214	**Tipo de solicitud:** *robot*
Layer 1 (LLM - robot)	00:00:142	**Comando:** seed, **Objeto:** pot4

7.4.3 Pruebas: IO

Prueba 6: Ajustar Temperatura

Comando Transcrito:

It's too hot inside. Lower the temperature to 20.5 Celsius degrees.

Respuesta

En la tabla 33 se muestra la respuesta de cada capa del árbol de decisiones al comando anterior:

Tabla 33: Respuesta de cada capa del árbol de decisiones para un comando de temperatura

Capa (Layer)	Tiempo de Respuesta	Respuesta
Layer 0 (LLM)	00:00:511	**Tipo de solicitud:** *io*
Layer 1 (LLM - io)	00:00:189	**Comando:** temperature, **Valor:** 20.5

Prueba 7: Ajustar Umbral de Luz

Comando Transcrito:

I want the light threshold to be 200.

Respuesta

En la tabla 34 se muestra la respuesta de cada capa del árbol de decisiones al comando anterior:

Tabla 34: Respuesta de cada capa del árbol de decisiones para un comando de luz

Capa (Layer)	Tiempo de Respuesta	Respuesta
Layer 0 (LLM)	00:00:422	**Tipo de solicitud:** *io*
Layer 1 (LLM - io)	00:00:331	**Comando:** light, **Valor:** 200

Prueba 8: Riego de la Planta 3

Comando Transcrito:

Water the plant 3.

Respuesta

En la tabla 35 se muestra la respuesta de cada capa del árbol de decisiones al comando anterior:

Tabla 35: Respuesta de cada capa del árbol de decisiones para un comando de riego

Capa (Layer)	Tiempo de Respuesta	Respuesta
Layer 0 (LLM)	00:00:981	**Tipo de solicitud:** *io*
Layer 1 (LLM - io)	00:00:242	**Comando:** irrigation, **Objeto:** 3

7.4.4 Pruebas: Info

Prueba 9: Consultar Temperatura en el Módulo

Comando Transcrito:

What is the temperature inside the module?

Respuesta

En la tabla 36 se muestra la respuesta de cada capa del árbol de decisiones al comando anterior:

Tabla 36: Respuesta de cada capa del árbol de decisiones para un comando de información sobre io

Capa (Layer)	Tiempo de Respuesta	Respuesta
Layer 0 (LLM)	00:00:773	**Tipo de solicitud:** *info*
Layer 1 (LLM - info)	00:00:336	**Comando:** sensor

Prueba 10: Estado del Robot

Comando Transcrito:

Tell me the Robot Status.

Respuesta

En la tabla 37 se muestra la respuesta de cada capa del árbol de decisiones al comando anterior:

Tabla 37: Respuesta de cada capa del árbol de decisiones para un comando de información del robot

Capa (Layer)	Tiempo de Respuesta	Respuesta
Layer 0 (LLM)	00:00:882	**Tipo de solicitud:** *info*
Layer 1 (LLM - info)	00:00:116	**Comando:** robot

7.4.5 Pruebas: Comandos implícitos

Prueba 11: Niveles de CO2

Comando Transcrito:

The CO2 levels inside the module are too high. I need you to do something about it.

Respuesta

En la tabla 38 se muestra la respuesta de cada capa del árbol de decisiones al comando anterior:

Tabla 38: Respuesta de cada capa del árbol de decisiones para un comando implícito de los IO

Capa (Layer)	Tiempo de Respuesta	Respuesta
Layer 0 (LLM)	00:00:320	**Tipo de solicitud:** *io*
Layer 1 (LLM - io)	00:00:406	**Comando:** CO2, **Valor:** 800

Prueba 12: Remover Planta 2

Comando Transcrito:

The second plant is too big, I need you to do something about it.

Respuesta

En la tabla 39 se muestra la respuesta de cada capa del árbol de decisiones al comando anterior:

Tabla 39: Respuesta de cada capa del árbol de decisiones para un comando implícito del robot

Capa (Layer)	Tiempo de Respuesta	Respuesta
Layer 0 (LLM)	00:00:218	**Tipo de solicitud:** *robot*
Layer 1 (LLM - robot)	00:00:122	**Comando:** takeout, **Objeto:** pot2

7.4.6 Conclusión

El asistente de voz basado en LLM ha demostrado ser preciso y eficiente en la gestión del sistema mecatrónico automatizado de cuidado de plantas. Su arquitectura modular, junto con una estrategia efectiva de ingeniería del prompt, garantiza respuestas rápidas y estructuradas. Las pruebas confirman su fiabilidad y capacidad para interpretar comandos, incluso en presencia de errores de transcripción. Dado su buen desempeño, se justifica su integración en el sistema, optimizando la interacción y el control automatizado.

8 Conclusiones

El desarrollo del **Sistema Mecatrónico Automatizado de Germinación de Semillas** ha representado un avance significativo en la automatización agrícola y control avanzado con LLMs, integrando tecnologías innovadoras para optimizar la eficiencia y reducir la intervención manual. A lo largo de este proyecto, se han abordado con éxito los principales desafíos en el diseño, implementación y validación del sistema, logrando cumplir con los objetivos planteados inicialmente.

8.1 Evaluación del Sistema

El sistema desarrollado ha demostrado ser funcional y eficiente en la automatización del proceso de germinación de semillas. Entre los principales logros destacan:

- **Automatización Integral del Proceso:** La integración del robot Mitsubishi Melfa RV-7FC-D, junto con la Raspberry Pi y un conjunto de sensores y actuadores, ha permitido una manipulación precisa de las macetas, un control ambiental eficiente y una siembra automatizada de alta precisión.
- **Sistema de Visión Artificial y Análisis de Imágenes:** La implementación de una cámara con análisis basado en GPT-4o ha facilitado la detección del estado de crecimiento de las plantas y la identificación temprana de enfermedades con gran precisión, reduciendo la necesidad de inspecciones manuales y optimizando la toma de decisiones.
- **Interfaz Humano-Máquina Avanzada:** Se ha desarrollado un asistente por voz basado en LLM, lo que permite una interacción en lenguaje natural con el sistema, mejorando la accesibilidad y facilidad de uso.
- **Implementación de un Gemelo Digital:** La sincronización en tiempo real entre el robot físico y su versión digital ha permitido realizar simulaciones y pruebas sin riesgos, optimizando la validación de estrategias de control.

8.2 Beneficios y Aportaciones

Este proyecto representa un **caso de éxito en la automatización agrícola**, ofreciendo diversas ventajas:

- **Eficiencia y Reducción de Costes en el Desarrollo de Funcionalidades con IA:** Los métodos de control desarrollados demuestran que es posible obtener funcionalidades avanzadas con inteligencia artificial sin requerir una gran inversión en el desarrollo de modelos propios.

- **Escalabilidad y Adaptabilidad:** La arquitectura modular del sistema permite su adaptación a distintos entornos, contextos y futuros desarrollos tecnológicos.
- **Ingeniería de Sistemas con IA:** Este proyecto no solo integra las tecnologías de inteligencia artificial más recientes, sino que también sirve como ejemplo de cómo incorporar estas tecnologías en sistemas ciberfísicos de manera efectiva.
- **Control de Sistemas Complejos con un Microcontrolador:** Se ha demostrado la viabilidad de controlar un sistema ciberfísico complejo con un microcontrolador de bajo coste, lo que puede inspirar a otros a explorar soluciones similares en diversos entornos industriales y académicos.

8.3 Futuras Líneas de Investigación

Este trabajo abre nuevas oportunidades para la investigación y el desarrollo en el ámbito de los demostradores industriales:

- **Uso de Reinforcement Learning para el Control de Robots:** Como línea de investigación futura, se plantea la incorporación de **Reinforcement Learning** en el control de robots. Esta es una tecnología emergente con gran potencial, y se busca explorar su integración en sistemas mecatrónicos de manera eficiente y accesible.
- **Inclusión de Control por Gestos:** Aprovechando el trabajo realizado por una compañera de equipo en su tesis de maestría, se prevé incorporar el control por gestos al sistema. Se espera que esta funcionalidad pueda combinarse con el control por voz para mejorar la interfaz hombre-máquina y hacerla más intuitiva.
- **Validación de Modelos LLM para el Control Industrial:** Dado que los modelos **LLM** son inherentemente no deterministas y operan como "cajas negras", su aplicación en entornos industriales críticos plantea interrogantes. En el futuro, se realizarán pruebas avanzadas para evaluar la viabilidad de estos modelos en escenarios de producción, con el objetivo de validar su uso mediante metodologías rigurosas.

8.4 Posibilidades de Implementación en Entornos Industriales y de Producción

Dado que el objetivo del proyecto es demostrar la integración de tecnologías emergentes en sistemas mecatrónicos, es fundamental analizar su potencial aplicación en entornos industriales:

- **Uso de IA Determinista:** Para garantizar la aplicabilidad en entornos industriales regulados, las soluciones de IA deben ser deterministas. Existen varias estrategias para lograrlo:
    - **Desarrollo de Modelos Propios:** En la industria, la creación de modelos transparentes y personalizables ha sido la estrategia más adoptada. Para facilitar esta transición, se han almacenado todas las entradas y salidas de los modelos actuales en bases de datos, permitiendo la generación de conjuntos de datos especializados para futuros entrenamientos.
    - **Validación de Modelos Existentes:** Aunque los modelos LLM actuales carecen de determinismo, se podría explorar su validación mediante pruebas formales. Dado el avance constante en inteligencia artificial y las crecientes regulaciones, es un área de investigación en constante evolución.
    - **Adaptación a Modelos Transparentes:** Se está trabajando en el desarrollo de modelos de IA más transparentes y deterministas. Si bien el equipo de este proyecto no está enfocado en el desarrollo de tales modelos, sí se contempla la posibilidad de adoptar estas soluciones a medida que se vuelvan viables para la industria.
- **Uso de Hardware Real-Time Industrial:** Aunque el sistema basado en Raspberry Pi es versátil, no cumple con los requisitos de tiempo real exigidos en entornos industriales. Para garantizar la fiabilidad del sistema, se podría considerar:
    - **PLC en un Sistema Distribuido:** Una opción clásica es utilizar un PLC para el control en tiempo real, complementado con un PC encargado de los cálculos complejos y la gestión del software.
    - **IPC (Industrial PC) como Alternativa Moderna:** Los IPC combinan la capacidad de un PLC con las prestaciones de un PC, permitiendo la implementación de estrategias avanzadas de ingeniería de software sin comprometer el tiempo real y el determinismo necesarios para el control de sensores y actuadores. En este caso, la Raspberry Pi podría ser reemplazada por un IPC, junto con la integración de un protocolo de comunicación en tiempo real para la interacción con el robot.

Con estos enfoques, se podría llevar este demostrador a un nivel de aplicación industrial, asegurando su viabilidad en entornos de producción reales.

8.5 Conclusiones Personales

La realización de este proyecto ha supuesto un gran crecimiento tanto a nivel personal como profesional. Haber desarrollado este trabajo en el entorno de una empresa ubicada en Múnich, Alemania, y dentro de un equipo internacional, me ha permitido mejorar significativamente mis habilidades sociales, comunicativas y de trabajo en equipo en un contexto multicultural.

Asumir el rol de *Product Owner* del proyecto me ha brindado una valiosa experiencia en la gestión de proyectos, planificación de tareas y coordinación de equipos. Esta responsabilidad me ha ayudado a desarrollar una mayor capacidad de organización, toma de decisiones y gestión eficiente del tiempo, habilidades fundamentales para mi futuro profesional.

Desde el punto de vista técnico, el proyecto me ha permitido consolidar y ampliar los conocimientos adquiridos durante el Grado en Ingeniería Electrónica Industrial y Automática. He profundizado en áreas como el diseño y documentación de circuitos eléctricos, la programación de robots y microcontroladores, así como el diseño de sistemas de comunicación. Además, he tenido la oportunidad de aprender nuevas tecnologías y herramientas punteras en el ámbito de la automatización y la inteligencia artificial, como el uso de modelos de lenguaje (LLM) para la toma de decisiones y el análisis de imágenes, el diseño de arquitecturas de software distribuidas o la implementación de gemelos digitales.

En conjunto, esta experiencia ha contribuido de forma muy positiva a mi formación integral como ingeniero y me ha preparado para enfrentar con mayor seguridad y madurez los retos del entorno laboral.

8.6 Conclusión Final

El desarrollo del **Sistema Mecatrónico Automatizado de Germinación de Semillas** ha logrado integrar de manera exitosa tecnologías avanzadas de robótica, inteligencia y visión artificiales, demostrando su viabilidad como solución innovadora en la automatización agrícola. A pesar de los desafíos encontrados, los resultados obtenidos validan el potencial de esta tecnología para mejorar la eficiencia y sostenibilidad en la producción agrícola. Este proyecto no solo representa un avance en el sector, sino que también sienta las bases para futuras innovaciones en la interacción entre la inteligencia artificial y los sistemas mecatrónicos.

Bibliografía

[1] ITQ GmbH. (s.f.). *Demostradores ITQ: Inspiración para la automatización del futuro*. https://www.itq.de/innovationen/demonstratoren/ (Último acceso al enlace: Marzo, 2025)

[2] Su, D., Qiao, Y., Jiang, Y., Valente, J., Zhang, Z., & He, D. (2023). Editorial: AI, sensors and robotics in plant phenotyping and precision agriculture, volume II. *Frontiers in Plant Science, 14*, 1215899. https://doi.org/10.3389/fpls.2023.1215899

[3] Kumar, P., & Ashok, G. (2021). Design and fabrication of smart seed sowing robot. *Materials Today: Proceedings, 39*(Part 1), 354–358. https://doi.org/10.1016/j.matpr.2020.07.432

[4] FarmBot Inc. (s.f.). *FarmBot Genesis*. https://farm.bot/pages/genesis (Último acceso al enlace: Marzo, 2025)

[5] Simon, L., Asiabanpour, B., & Summers, M. (2023). Improving automated plant seeding through design and development of automated seeder and route optimization. En *2023 Congress in Computer Science, Computer Engineering, & Applied Computing (CSCE)* (pp. 922–928). IEEE. https://doi.org/10.1109/CSCE60160.2023.00156

[6] Wu, Z., Shu, P., Li, Y., Li, Q., Liu, T., & Li, X. (2024). Robot control via natural instructions empowered by large language model. En R. Vinjamuri (Ed.), *Discovering the frontiers of human-robot interaction* (pp. 437–457). Springer. https://doi.org/10.1007/978-3-031-66656-8_19

[7] Zhang, C., Chen, J., Li, J., Peng, Y., & Mao, Z. (2023). Large language models for human–robot interaction: A review. *Biomimetic Intelligence and Robotics, 3*(4), 100131. https://doi.org/10.1016/j.birob.2023.100131

[8] Papakitsos, E., Giachos, I., Papoutsidakis, M., Piromalis, D., & Kaminaris, S. D. (2020). A contemporary survey on intelligent human-robot interfaces focused on natural language processing. *International Journal of Advanced Research in Artificial Intelligence (IJARAI), 8*(6), 1–20. https://www.researchgate.net/publication/342720301_A_Contemporary_Survey_on_Intelligent_Human-Robot_Interfaces_Focused_on_Natural_Language_Processing

[9] Schwaber, K., & Sutherland, J. (s.f.). *The Scrum Guide: The definitive guide to Scrum: The rules of the game*. Scrum Guides. https://scrumguides.org/ (Último acceso al enlace: Marzo, 2025)

[10] International Electrotechnical Commission. (2021). IEC 60445:2021 – Basic and safety principles for man-machine interface, marking and

identification – Identification of equipment terminals, conductor terminations and conductors.

[11] International Electrotechnical Commission. (2021). IEC 60364-5-52:2022 – Low-voltage electrical installations – Part 5-52: Selection and erection of electrical equipment – Wiring systems.

[12] Mitsubishi Electric Corporation. (2023). *RV-4F/7F/13F/20F/35F/50F/70F series robot arm: Setup and maintenance instruction manual (BFP-A8935-AE)* [Manual de usuario]. https://dl.mitsubishielectric.com/dl/fa/document/manual/robot/bfp-a8935/bfp-a8935ae.pdf (Último acceso al enlace: Marzo, 2025)

[13] Mitsubishi Electric Corporation. (2023). *RV-4F-D/7F-D/13F-D/20F-D/35F-D/50F-D/70F-D series: Standard specifications manual – For CR750-D/CR751-D/CR760-D controllers (BFP-A8931-AJ)* [Manual de usuario]. https://dl.mitsubishielectric.com/dl/fa/document/manual/robot/bfp-a8931/bfp-a8931aj.pdf (Último acceso al enlace: Marzo, 2025)

[14] Mitsubishi Electric Corporation. (2024). *CR750/CR751/CR800/CR860 series controller: Ethernet function instruction manual (BFP-A3379-G)* [Manual de usuario]. https://dl.mitsubishielectric.com/dl/fa/document/manual/robot/bfp-a3379/bfp-a3379g.pdf (Último acceso al enlace: Marzo, 2025)

[15] Mitsubishi Electric Corporation. (2021). *CR750-D/CR751-D/CR760-D controller: Setup, basic operation, and maintenance (BFP-A8867-AF)* [Manual de usuario]. https://dl.mitsubishielectric.com/dl/fa/document/manual/robot/bfp-a8867/bfp-a8867af.pdf (Último acceso al enlace: Marzo, 2025)

[16] Mitsubishi Electric Corporation. (2023). *CR750/CR751/CR760 series controller: Instruction manual – Detailed explanations of functions and operations (BFP-A8869-AE)* [Manual de usuario]. https://dl.mitsubishielectric.com/dl/fa/document/manual/robot/bfp-a8869/bfp-a8869ae.pdf (Último acceso al enlace: Marzo, 2025)

[17] Computing For All. (2024, Julio, 04). *GPT-4o for Video Summarization: A Computer Vision Project in Python* [Video]. YouTube. https://www.youtube.com/watch?v=h07cAyLrL6w (Último acceso al enlace: Marzo, 2025)

[18] ML Explained (2023, Noviembre, 08) *A Guide on How To Use GPT-4 Vision API | System Prompt and User Prompt Explained* [Video]. YouTube.

https://www.youtube.com/watch?v=5O-1SDxiwkM (Último acceso al enlace: Marzo, 2025)

[19] OpenAI. (2024). *Vision*. OpenAI Platform Documentation. https://platform.openai.com/docs/guides/vision (Último acceso al enlace: Marzo, 2025)

[20] Ollama. (2024). *Ollama API reference*. GitHub. https://github.com/ollama/ollama/blob/main/docs/api.md (Último acceso al enlace: Marzo, 2025)