ESCUELA SUPERIOR POLITECNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

“Diseño e implementación de un prototipo a escala de una planta de fluidos para el laboratorio de sistemas de control y el desarrollo de prácticas de laboratorio”

PROYECTO INTEGRADOR

Previo a la obtención del Título de:

Ingeniero en Electrónica y Automatización

Presentado por:

Saúl Francisco Carvallo Velasco

Jonathan Francisco Zambrano Reyna

GUAYAQUIL – ECUADOR

Año: 2023

DEDICATORIA

Este proyecto va dedicado a mis amados padres, Patricia del Carmen Velasco Alvarado y Francisco Xavier Carvallo Castillo, que con su amor, apoyo y sacrificio han sido la fuerza detrás de este logro. A mi querida tía Leticia Velasco Alvarado, que me inspiró e iluminó en distintos momentos de la trayectoria de este camino dejando muchas enseñanzas.

A mis amigos, que estuvieron conmigo en las buenas y en las malas incentivándome a que nunca me rinda.

Con inmensa gratitud,

-Saúl Francisco Carvallo Velasco

DEDICATORIA

Este proyecto va dedicado a mi querido padre, Alfredo Zambrano Alcívar, quien con su amor, sacrificio y dedicación sembró en mí el valor de la educación, su lucha incesante por mi formación es el legado más preciado que me ha dejado, que, aunque ya no este físicamente, su presencia y enseñanzas perduraran en cada logro alcanzado. A mi querida madre Nancy Reyna Vera, que me motivo en cada obstáculo que se me presentó a lo largo de este camino, su apoyo total y su inagotable amor han sido el motor en los momentos más difíciles.

Finalmente, agradezco al Dr. Yonny Demera y a su esposa Sonia Zambrano que me cobijaron en su hogar como un hijo más y nunca me dejaron solo en este trayecto tan importante,

Con infinita gratitud.

- Jonathan Francisco Zambrano Reyna

AGRADECIMIENTOS

Primero quiero agradecer a mis amados padres, Patricia del Carmen Velasco Alvarado y Francisco Xavier Carvallo Castillo, que nunca dejaron de creer en mí y fueron un pilar fundamental para este logro.

A mis amigos más cercanos que no me dejaron solo en ningún momento, dándome la mano cada que me caía.

Con inmensa gratitud,
-Saúl Francisco Carvallo Velasco

AGRADECIMIENTOS

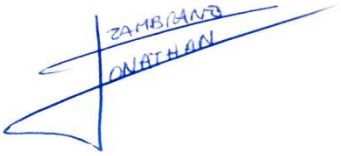
Quiero expresar mi profunda gratitud al Dr Yonny Demera y su esposa Sra Sonia Zambrano por su apoyo incondicional en todo momento, que en ningún momento dejaron de creer en mis capacidades. A mi padre Alfredo Zambrano por todo su sacrificio para que nunca me faltara nada y que siempre me motivo a superarme a nivel personal y profesional.

-Jonathan Francisco Zambrano Reyna

DECLARACIÓN EXPRESA

“Los derechos de titularidad y explotación, nos corresponde conforme al reglamento de propiedad intelectual de la institución; *Saúl Francisco Carvallo Velasco* y *Jonathan Francisco Zambrano Reyna* damos nuestro consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual”

Saúl Francisco Carvallo Velasco



Jonathan Francisco Zambrano Reyna

EVALUADORES

EFREN VINICIO HERRERA MUENTES

Digitally signed by EFREN VINICIO HERRERA MUENTES
Date: 2024.02.11 20:28:03 -05'00'

Efrén Vinicio Herrera Muentes

PROFESOR DE LA MATERIA

Firmado electrónicamente por:
RONALD DAVID SOLIS MESA

Ronald David Solís Mesa

PROFESOR TUTOR

RESUMEN

Este proyecto tiene como objetivo desarrollar un prototipo a escala de una planta de fluidos para el laboratorio de sistemas de control en la Escuela Superior Politécnica del Litoral. El prototipo, diseñado con elementos accesibles, busca superar la falta de recursos para enseñar sobre plantas de fluidos en el laboratorio. La implementación de esta planta ofrece a los estudiantes la oportunidad de aplicar sus conocimientos teóricos en sistemas de control en un entorno práctico, utilizando diversos tipos de controladores.

El proceso incluyó el diseño mecánico de la estructura utilizando Inventor, con pruebas de seguridad para garantizar la estabilidad. Se elaboró un diseño eléctrico/electrónico en Proteus para el circuito que integra la Raspberry Pi con la planta de fluidos, seguido del montaje de la estructura y los componentes. Se programaron controladores ON/OFF, PID y Fuzzy, y se compararon los resultados para determinar que el controlador Fuzzy fue el más eficiente, con un tiempo de estabilización superior al PID. La planta resultante se demostró apta para diversas prácticas de laboratorio en sistemas de control, y se documentaron los resultados a través de un HMI en MATLAB, con los datos almacenados en la nube en ThingSpeak para futuras aplicaciones prácticas.

En conclusión, el proyecto logró desarrollar una planta de fluidos funcional, con el controlador Fuzzy destacándose como la mejor opción para aplicaciones prácticas en el entorno educativo.

Palabras clave: Controladores, PID, Fuzzy, MATLAB, ThingSpeak.

ABSTRACT

This project aims to develop a scaled prototype of a fluid plant for the control systems laboratory at Escuela Superior Politecnica del Litoral. The prototype, designed with readily available elements, addresses the lack of resources for teaching fluid plants in the laboratory. Implementing this plant provides students with the opportunity to apply their theoretical knowledge in control systems in a practical environment, using various types of controllers.

The process included the mechanical design of the structure using Inventor, with safety tests to ensure stability. An electrical/electronic design in Proteus was created for the circuit integrating the Raspberry Pi with the fluid plant, followed by the assembly of the structure and components. ON/OFF, PID, and Fuzzy controllers were programmed, and results were compared to determine that the Fuzzy controller was the most efficient, with a stabilization time superior to PID. The resulting plant was proven suitable for various control systems laboratory practices, and results were documented through an HMI in MATLAB, with data stored in the cloud on ThingSpeak for future practical applications.

In conclusion, the project successfully developed a functional fluid plant, with the Fuzzy controller standing out as the best choice for practical applications in the educational environment.

Keywords: Controllers, PID, Fuzzy, MATLAB, ThingSpeak.

ÍNDICE GENERAL

CAPÍTULO 1 1
1. INTRODUCCIÓN 1
1.1 DESCRIPCIÓN DEL PROBLEMA 1
1.2 JUSTIFICACIÓN DEL PROBLEMA 2
1.3 OBJETIVOS 3
1.3.1 Objetivo General 3
1.3.2 Objetivos Específicos 3
1.4 MARCO TEÓRICO 4
1.4.1 Automatización y Control de Procesos 4
1.4.1.1 Definición y Conceptos Fundamentales de Automatización 4
1.4.1.2 Importancia de la Automatización en la industria 4
1.4.1.3 Control de Procesos y sus Aplicaciones 4
1.4.2 Plantas de Fluidos 5
1.4.2.1 Concepto y Características de las Plantas de Fluidos 5
1.4.2.2 Tipos de Plantas de Fluidos y sus Aplicaciones 6
1.4.2.3 Importancia de un Diseño Eficiente en Plantas de Fluidos 6
1.4.2.4 Variables de Control en Sistemas de Fluidos 6
1.4.2.5 Estrategias de Control Utilizadas en Plantas de Fluidos 7
1.4.3 Raspberry Pi 4 10
1.4.3.1 Introducción a la Raspberry 4 10
1.4.3.2 Características y Capacidades de la Raspberry Pi 4 10
1.4.3.3 Programación y Configuración de la Raspberry Pi 4 11
1.4.4 MATLAB y Simulink 12
1.4.4.1 Modelado y Simulación de Sistemas con Simulink 13
1.4.4.2 Integración de MATLAB y Simulink en Proyectos de Control 13
1.4.5.1 Concepto y Cómo funciona el Internet Industrial de las Cosas: 14

1.4.5.2 Conexión a la Nube y Recolección de Datos en Proyectos IIoT 15
1.4.5.3 Beneficios del IIoT en Sistemas de Control 15
CAPÍTULO 2 16
2. METODOLOGÍA 16
2.1 Selección de Componentes 16
2.2 Diseño mecánico de la planta. 17
2.2.1 Modelo de la plataforma de la planta 18
2.2.2 Modelo del montaje de los elementos de la planta 18
2.3 Dimensionamiento de los componentes y equipos 19
2.3.1 Dimensionamiento mecánico 19
2.3.2 Diseño eléctrico-electrónico en proteus 22
2.3.3 Dimensionamiento eléctrico 22
2.3.3.1 Fuente 22
2.3.3.2 Raspberry Pi 23
2.3.3.3 Pantalla Raspberry Pi de 7 pulgadas IPS 1024 x 600 HDMI 23
2.3.3.4 Bomba de Agua 24
2.3.3.5 Relé SRD – 5VDC – SL – C 25
2.3.3.6 Sensor ultrasónico HC-SR04 26
2.3.3.7 Electroválvula 27
2.3.4 Montaje de los equipos 28
2.3.5 Diagrama de flujo del controlador PID 28
2.3.5.1 Diagrama de flujo del controlador Fuzzy 30
2.3.5.2 Reglas del controlador Fuzzy 31
2.3.5.3 Ingreso a la raspberry 31
2.3.5.4 Programación realizada en visual studio code dentro de la raspberry 32
2.3.5.5 Código para el control de nivel 33

2.3.5.6 Código para la lectura en la raspberry 34
CAPÍTULO 3 35
3 RESULTADOS Y ANÁLISIS 35
3.1 Fabricación de la estructura metálica y montaje de los equipos 35
3.2 Modelo de la función de transferencia en Matlab 39
3.2.1 función de transferencia 39
3.2.1.1 Grafica de función de transferencia 40
3.2.2 Sintonización del controlador 40
3.2.3 Validación del modelo 41
3.3 Graficas obtenidas con controlador PID 42
3.4 Graficas obtenidas con controlador Fuzzy 44
3.5 Interfaz obtenida en la pantalla 46
3.6 Programación del controlador PID en Python 47
CAPÍTULO 4 56
4 CONCLUSIONES Y RECOMENDACIONES 56
4.1 Conclusiones 56
4.2 Recomendaciones 57
BIBLIOGRAFÍA 70

ABREVIATURAS

HMI Interfaz Hombre Máquina

PID Proporcional Integral Derivativo

AC Corriente Alterna

DC Corriente Continua

SIMBOLOGÍA

V	Voltaje
A	Corriente
W	Potencia
Kg	Kilogramos
mA	Miliamperios
mV	Milivoltios
px	Pixeles
Pa	Pascales
mPa	Milipascales
N	Newtons
Mb	Megabytes
Gb	Gigabytes
S	Segundos
µS	Microsegundos
in	Pulgadas
cm	Centímetros
m	Metros
Hz	Hertz
KHz	Kilohertz
°C	Centígrados

ÍNDICE DE FIGURAS

Figura. 1.1 Controlador Fuzzy 9
Figura. 1.2 *Tarjeta Raspberry Pi 4* 12
Figura. 1.3 Ilustración de enlace entre la nube y la industria 15
Figura. 2.1 Diagrama de flujo para la solución 16
Figura. 2.2 Modelo de la plataforma de la planta embebida. 18
Figura. 2.3 Vista representativa con los elementos electrónicos montados en campo. 18
Figura. 2.4 Fuerza aplicada en cada sección de la estructura mecánica. 19
Figura. 2.5 Factor de seguridad con umbral mínimo de 6.46 y máximo de 15. 20
Figura. 2.6 Altura del prototipo considerando tanque. 21
Figura. 2.7 Ancho del prototipo final. 21
Figura. 2.8 Diagrama general de conexiones, eléctricas y electrónicas. 22
Figura. 2.9 Fuente 12VDC 23
Figura. 2.10 Pantalla LCD Raspberry Pi 24
Figura. 2.11 Bomba de agua 25
Figura. 2.12 Relé 5VDC 25
Figura. 2.13 Sensor ultrasónico 26
Figura. 2.14 Electroválvula de uso industrial 27
Figura. 2.15 Prototipo de la planta implementado en diseño mecánico validado. 28
Figura. 2.16 Diagrama de flujo del controlador PID del prototipo 29
Figura 2.17 Diagrama de flujo del controlador Fuzzy del prototipo 30
Figura 2.18 Reglas del controlador Fuzzy 31
Figura. 2.19 Interfaz de inicio de la Raspberry. 31
Figura. 2.20 Configuración del servidor a utilizar. 32
Figura. 2.21 Pantalla de inicio de la Raspberry. 32
Figura. 2.22 Visual Studio Code embebido dentro de la Raspberry. 32
Figura. 2.23 Código para la lectura en la raspberry 34
Figura. 3.1 Planchas de acero galvanizado para el prototipo base. 35
Figura. 3.2 Tubos huecos con diámetro de 1 pulgada 35
Figura. 3.3 Estructura metálica finalizada 36
Figura. 3.4 Pruebas de montaje de los elementos 36
Figura. 3.5 Aplicación de capa de pintura a la estructura metálica 37

Figura. 3.6 Montaje de los elementos de la planta de fluidos 37
Figura. 3.7 Montaje de los elementos de la planta de fluidos 38
Figura. 3.8 Montaje de los elementos de la planta de fluidos 38
Figura. 3.9 Modelo de identificación de la planta prototipada. 39
Figura. 3.10 Grafica comparativa de datos experimentales y modelo identificado ... 40
Figura 3.11 Control de nivel usando controlador PID con error de 0.231146 42
Figura 3.12 Control de nivel usando controlador PID con error de 0.54747 43
Figura 3.13 Control de nivel usando controlador PID con error de 0.346699 43
Figura 3.14 Control de nivel usando controlador Fuzzy con error de 0.645679 44
Figura 3.15 Control de nivel usando controlador Fuzzy con error de 0.152663 44
Figura 3.16 Control de nivel usando controlador Fuzzy con error de 0.182273 45
Figura 3.17 Control de nivel usando controlador Fuzzy con error de 0.25423 45
Figura. 3.18 Interfaz de usuario del prototipo embebido utilizando Qt 46
Figura 3.19 Interfaz de usuario del prototipo embebido utilizando Matlab 46
Figura 3.20 Conexión a la nube con IoT 53
Figura 3.21 Dashboard creado en ThingSpeak 54
Figura 3.22 Datos almacenados del volumen del tanque y setpoint 54
Figura 3.23 Datos del error y volumen 55
Figura 3.24 Datos del error y litros de agua en el tanque 55

ÍNDICE DE TABLAS

Tabla 2.1: Listado de componentes 17
Tabla 2.2 Datos técnicos de la fuente 23
Tabla 2.3 Datos técnicos de la pantalla raspberry pi de 7 pulgadas 24
Tabla 2.4 Datos técnicos de la bomba de agua 25
Tabla 2.5 Datos técnicos del relé 26
Tabla 2.6 Datos técnicos del sensor ultrasónico 26
Tabla 2.7 Datos técnicos de la electroválvula 27
Tabla 3.1 Validación del modelo 41
Tabla 3.2 Comparativa entre programación Python vs Matlab 53

ÍNDICE DE GRÁFICOS

Grafica. 3.1 Validación del modelo 41
Grafica 3.2 Valores medidos 42

CAPÍTULO 1

1. INTRODUCCIÓN

Este proyecto de investigación explora en detalle el campo de las plantas de fluidos, incluyendo la selección de componentes y materiales, la automatización de procesos y la optimización de sistemas para mejorar la eficiencia energética. El núcleo de este trabajo se basa en el desarrollo de un prototipo de planta de fluidos, que servirá como un ejemplo concreto de cómo aplicar los principios teóricos en la práctica. A través de este prototipo, se podrán evaluar soluciones innovadoras en la gestión de fluidos y explorar los beneficios de la automatización y la elección adecuada de materiales.

A lo largo de las siguientes secciones, se abordarán en detalle los fundamentos de la mecánica de fluidos, los componentes y equipos relevantes, las consideraciones de diseño y las normativas aplicables. Además, se presentarán ejemplos concretos de controladores embebidos como lo son PID y Fuzzy dentro del control clásico.

1.1 DESCRIPCIÓN DEL PROBLEMA

En la Escuela Superior Politécnica del Litoral (ESPOL), se identifica una carencia significativa en el Laboratorio de Sistemas de Control: la ausencia de una planta de fluidos funcional e implementada a nivel físico. La falta de un sistema de este tipo limita la capacidad de los estudiantes y profesionales para investigar y experimentar con tecnologías relacionadas con el manejo de fluidos en un entorno de aprendizaje práctico.

El Laboratorio de Sistemas de Control es un componente esencial de la formación de ingenieros en electrónica y automatización, donde los estudiantes adquieren habilidades prácticas en el diseño y operación de sistemas de control en un entorno de laboratorio. Sin embargo, la carencia de una planta de fluidos, en este caso limita la capacidad de los estudiantes para explorar y aplicar conceptos de control en sistemas que implican el manejo de fluidos, un área esencial en numerosos campos de la ingeniería, desde la automatización industrial hasta la gestión de recursos hídricos.

Esta ausencia limita aún más la capacidad de la ESPOL para llevar a cabo investigaciones y proyectos relacionados con sistemas de control en el ámbito de la ingeniería de fluidos. No tener un sistema de prácticas y experimentación relacionado

con fluidos hace que sea difícil para los estudiantes y profesionales de la institución explorar nuevas tecnologías y metodologías en este campo crítico.

1.2 JUSTIFICACIÓN DEL PROBLEMA

La ausencia de una planta de fluidos en el Laboratorio de Sistemas de Control de la Escuela Superior Politécnica del Litoral (ESPOL) es un asunto que requiere atención y enfoque inmediato. Esta carencia se fundamenta en varias razones clave que destacan la importancia de abordar este problema.

Una de ellas es la relevancia en la formación académica, ya que la automatización y el control de sistemas que involucran fluidos son esenciales en numerosos campos de la ingeniería, desde la industria hasta la gestión de recursos hídricos y la biotecnología. La formación de ingenieros en electrónica y automatización debe reflejar la realidad de las aplicaciones industriales y científicas. La falta de una planta de fluidos limita la capacidad de los estudiantes para comprender y aplicar conceptos fundamentales en sistemas que gestionan fluidos, lo que resulta en una brecha entre la teoría y la práctica.

Además, la carencia en la Experimentación práctica es un problema significativo. La teoría sola no es suficiente para formar ingenieros competentes en el control y la automatización de sistemas de fluidos. Los estudiantes requieren experiencia práctica en la construcción, operación y control de sistemas reales que involucran fluidos. La falta de una planta de fluidos impide la experimentación práctica y la aplicación de los principios teóricos, lo que afecta negativamente la calidad de la educación y la preparación de los futuros ingenieros.

Otro aspecto relevante es la limitación en la investigación y desarrollo. La ausencia de una planta de fluidos también obstaculiza la capacidad de la ESPOL para llevar a cabo investigaciones en sistemas de control relacionados con fluidos y proyectos que requieren experimentación en un entorno de laboratorio. Esto limita la capacidad de la institución para innovar en áreas de investigación de importancia crítica y para colaborar con la industria en soluciones prácticas.

Finalmente, el desarrollo de competencias profesionales se ve comprometido por la falta de una planta de fluidos en el Laboratorio de sistemas de control. Los ingenieros que se gradúan de la ESPOL deben estar preparados para enfrentar los desafíos de la industria y contribuir al desarrollo económico y tecnológico del país. La carencia de

esta instalación limita la capacidad de la institución para desarrollar competencias profesionales y prácticas en sus estudiantes, lo que puede afectar su empleabilidad y contribución a la sociedad. En resumen, la ausencia de una planta de fluidos en el laboratorio de sistemas de control de la ESPOL tiene un impacto significativo en la formación académica, la experiencia práctica, la investigación y el desarrollo, así como en el desarrollo de competencias profesionales de los estudiantes, lo que justifica la necesidad urgente de diseñar e implementar un prototipo a escala de una planta de fluidos en dicho laboratorio.

1.3 OBJETIVOS

1.3.1 Objetivo General

Implementar un prototipo de planta de fluidos a escala en el laboratorio de sistemas de control para contribuir a las prácticas de la materia, promoviendo la adquisición de habilidades de sintonización de controladores.

1.3.2 Objetivos Específicos

- Investigar los diferentes tipos de plantas de fluidos en los cuales permita controladores y modelos matemáticos de primer orden.
- Desarrollar un prototipo para la sintonización de controladores PID y Fuzzy a través del ingreso de sus constantes integral Ki, proporcional Kp,derivativa Kd.
- Identificar una planta para la obtención del modelo matemático del sistema.
- Establecer el controlador PID para controlar la planta.
- Definir un controlador fuzzy para el control de la planta a través del uso de Matlab.
- Implementar sistemas de visualización a través de una dashboard con tecnología IoT.

1.4 MARCO TEÓRICO

1.4.1 Automatización y Control de Procesos

1.4.1.1 Definición y Conceptos Fundamentales de Automatización

La automatización es la encargada de emplear sistemas, tecnología y controladores para ejecutar labores y procedimientos de manera automática, omitiendo la intervención directa de los seres humanos.

Dentro de un entorno industrial, la automatización desempeña un papel fundamental en la mejora de la eficacia y la calidad de las operaciones.

1.4.1.2 Importancia de la Automatización en la industria

La importancia de la automatización en una industria se da por los siguientes puntos:

- Garantizar el incremento de la producción.
- Mejorar los resultados de las operaciones que realiza la industria.
- Reducción de costos estandarizando el flujo de procesos.
- Reducción de riesgos para los trabajadores.
- Monitoreo en tiempo real y toma de decisiones.
- Innovación y desarrollo tecnológico.

1.4.1.3 Control de Procesos y sus Aplicaciones

“El control de procesos toma en cuenta la medición y el análisis de las variables que determinan el funcionamiento de un proceso, así como la toma de decisiones y la ejecución de acciones de control para gobernar dicho proceso. ” (Arbildo Lopez, 2011)

Las diversas áreas donde se puede aplicar el control de procesos son las siguientes:

Manufactura: Garantizando la consistencia y calidad en la producción de bienes como fármacos, químicos, dispositivos electrónicos, etc.

Plantas de energía: Regulando la producción y distribución de energía eléctrica en centrales eléctricas para satisfacer la demanda.

Tratamiento de aguas y aguas residuales: Asegurando la purificación del agua potable y gestionando eficientemente las aguas residuales en plantas de tratamiento.

Agricultura: Administrando la irrigación, aplicación de fertilizantes y pesticidas, contribuyendo en el monitoreo de cultivos.

Salud y Ciencias de la vida: Regulando las condiciones de laboratorio y sistemas de administración de medicamentos en aplicaciones médicas.

El control de procesos impacta de manera significativa en diversas industrias, mejorando la eficiencia y la calidad de su producción.

1.4.1.4 Microcontroladores y su rol en la automatización

Los microcontroladores tienen un papel esencial en la automatización de procesos al ofrecer una solución integrada y programable para supervisar y controlar operaciones específicas.

Sus componentes principales son:

CPU: Sus siglas significan “Unidad Central de Procesamiento” y es el cerebro del controlador programable, encargado de ejecutar las instrucciones programadas para controlar el proceso.

Memoria: Es la encargada de almacenar el programa y los datos necesarios para el control del sistema.

Periféricos de entradas y salidas (E/S): Las entradas son señales del entorno (Sensores), y las salidas son las señales enviadas al sistema (Actuadores). Están encargadas de facilitar la interacción del controlador con el proceso.

Interfaces de comunicación: Permiten la conexión y comunicación con otros dispositivos, tales como, buses de datos, UART, SPI, I2C, entre otros.

El rol que desempeñan en la automatización es el de controlar procesos específicos en tiempo real, ya sea la regulación de temperatura, monitoreo de niveles, gestión de motores, entre otros.

1.4.2 Plantas de Fluidos

1.4.2.1 Concepto y Características de las Plantas de Fluidos

El concepto de una planta de fluido se ve ligado a sistemas diseñados en al ámbito académico para el estudio y la experimentación en la manipulación y control de fluidos, como gases y líquidos. Estás plantas se dan en entornos académicos y de investigación en ingeniería, debido a estas condiciones practicas se involucran control y sistemas de flujo.

Las principales características que podemos encontrar en estas plantas son:

Manipulación de fluidos, este tipo de plantas permite la interacción con distintos fluidos como lo son agua, aceite, gases, etc.

Equipamiento especifico y componentes, estas plantas de fluidos están equipadas con una serie de componentes y dispositivos especializados, como bombas, válvulas, electroválvulas, tuberías, sensores de presión, medidores de caudal y actuadores. Estos componentes permiten al usuario simular entornos y controlar diferentes condiciones de presión y flujo.

Control y automatización, la capacidad de poder controlar y automatizar los procesos de flujo y las condiciones operativas como lo es los índices de desempeño dentro de la teoría de control es fundamental. Esto permite la realización de experimentos controlados y la práctica de estrategias de control en tiempo real.

1.4.2.2 Tipos de Plantas de Fluidos y sus Aplicaciones

Planta de agua potable, están plantas simulan todo lo relacionado al tratamiento de aguas y desalinización.

Planta de fluidos térmicos, estás son muy utilizadas para la generación de energía térmica y el análisis de transferencia de calor.

Planta de refrigeración, estas plantas replican sistemas de aires acondicionados y de refrigeración en espacios confinados.

Planta de control de flujo y válvulas, Utilizadas para estudiar sistemas de control de flujo y válvulas en diversos contextos industriales. Son esenciales para la formación en control automático y sistemas de automatización.

1.4.2.3 Importancia de un Diseño Eficiente en Plantas de Fluidos

La importancia de un diseño eficiente es fundamental en plantas de fluidos, al momento de su operación es importante analizar radica en su impacto en la práctica, el rendimiento y la seguridad de los sistemas que involucran fluidos, tales como son, bombas, válvulas, sensores y actuadores en donde un diseño eficiente no solo beneficia a nivel práctico, sino que también tiene implicaciones económicas en cuanto al diseño ingenieril que se le dé en base a objetivos educativos presentes.

1.4.2.4 Variables de Control en Sistemas de Fluidos

- **Caudal (Flujo):** El caudal se refiere a la cantidad de fluido que pasa por una sección de un conducto o tubería en un período de tiempo específico. Es una variable fundamental en sistemas de tuberías y conductos, ya que controlar el

caudal es esencial para garantizar un flujo adecuado y evitar obstrucciones o desbordamientos.

- **Presión:** La presión se refiere a la fuerza por unidad de área que ejerce el fluido en las paredes del sistema. Controlar la presión es esencial para evitar situaciones de sobre o supresión que podrían dañar equipos o comprometer la seguridad.
- **Temperatura:** La temperatura del fluido es una variable crítica en sistemas de calefacción, refrigeración y procesos químicos. El control de la temperatura garantiza que el fluido se mantenga dentro de un rango deseado para lograr los objetivos del proceso.
- **Nivel:** El nivel se refiere a la altura del fluido en un tanque o contenedor. Controlar el nivel es esencial para evitar desbordamientos o caídas en el suministro de fluido, lo que podría tener consecuencias negativas.
- **Velocidad del Motor o Bomba:** En sistemas que utilizan motores o bombas para mover fluidos, controlar la velocidad de estos dispositivos es esencial para ajustar el caudal y la presión del fluido.
- **Válvulas y Posición de Actuadores:** Las válvulas y actuadores son componentes clave en sistemas de fluidos. Controlar su apertura, cierre y posición es importante para regular el flujo y la dirección del fluido.
- **Densidad:** La densidad del fluido es relevante en aplicaciones como la medición de flujo de líquidos y gases. Controlar la densidad es necesario para obtener mediciones precisas.

1.4.2.5 Estrategias de Control Utilizadas en Plantas de Fluidos

1.4.2.5.1 Control ON/OFF

"La salida del controlador ON-OFF, o de dos posiciones, solo puede cambiar entre dos valores al igual que dos estados de un interruptor. El controlador no tiene la capacidad para producir un valor exacto en la variable controlada para un valor de referencia dado pues el controlador produce una continua desviación del valor de referencia." (Katsuhiko, 2010)

1.4.2.5.2 Control PID

“EL control bajo PID, es un mecanismo de genérico y para procesos cuyas dinámicas suelen representarse linealmente sobre una realimentación de bucle cerrado, ampliamente usado en la industria para el control de sistemas. El PID es un sistema al que le entra un error calculado a partir de la salida deseada menos la salida obtenida y su salida es utilizada como entrada en el sistema que queremos controlar. El controlador intenta minimizar el error ajustando la entrada del sistema.” (Astrom, 2000)

“Los controladores PID se consideran uno de los bucles de control tradicionales más importantes que se utilizan en la mayoría de las aplicaciones industriales, debido a su estructura simple y sus ventajas relacionadas con la respuesta transitoria y el error de estado estacionario. Estos controladores requieren el ajuste de sus parámetros de acuerdo con la aplicación donde se vuelven constantes a través del proceso de control, lo que hace que estos controladores no puedan controlar cuando ocurre una perturbación en el sistema.” (Ibrahim & Al Akkad, 2016)

1.4.2.5.3 Lógica Fuzzy

“La lógica difusa es una lógica que tiene muchos valores. A diferencia del sistema de lógica binaria, aquí el razonamiento no es nítido, sino que es aproximado y tiene un límite impreciso. El sistema basado en lógica difusa lleva a cabo el proceso de toma de decisiones mediante la incorporación del conocimiento humano en el sistema. El sistema de inferencia difusa es la unidad principal de un sistema de lógica difusa. La toma de decisiones es una parte importante de todo el sistema. El sistema formula las reglas adecuadas y, basándose en estas reglas, se toman las decisiones. Todo este proceso de toma de decisiones es principalmente la combinación de conceptos de conjunto difuso, normas IF-THEN difusas y razonamiento difuso. El sistema de inferencia difusa hace uso de las declaraciones IF-THEN y con la ayuda de los conectores presentes (como OR y AND), se construyen las reglas de decisión necesarias. El sistema básico de inferencia difusa puede tomar entradas borrosas o entradas nítidas dependiendo del proceso y sus salidas, en la mayoría de los casos, son conjuntos difusos.” (Amllick, 2011)

1.4.2.5.4 Control Fuzzy

"En 1974 Mamdani desarrolló un procedimiento de deducción difusa basado en las reglas sugeridas por Zadeh y se usaron para controlar las plantas dinámicas. Después de eso, los primeros controladores de lógica difusa fueron diseñados para manejar sistemas complejos que son difíciles de modelar matemáticamente. Estos FLC (Controlador Lógico Fuzzy), de controladores lógicos difusos requieren comprender el comportamiento de los sistemas de control y formar la experiencia humana con reglas difusas." (Ibrahim & Al Akkad, 2016)

"El controlador de lógica difusa emplea un método que transforma el saber especializado en una estrategia para el control automático. La lógica difusa gestiona información aproximada de manera sistemática, siendo idónea para el manejo de sistemas no lineales y la representación de sistemas complejos. Estos sistemas difusos se utilizan para modelar escenarios donde la precisión del modelo es imprecisa o donde la ambigüedad y la vaguedad son habituales, tal como se ilustra en la Figura 1.1, donde un conjunto de reglas difusas opera como un mecanismo de decisión para ajustar los efectos de ciertos estímulos del sistema. "La base de reglas refleja el conocimiento experto humano, expresado como variables lingüísticas, mientras que las funciones de membresía representan la interpretación experta de esas variables." (Padhee, 2011)

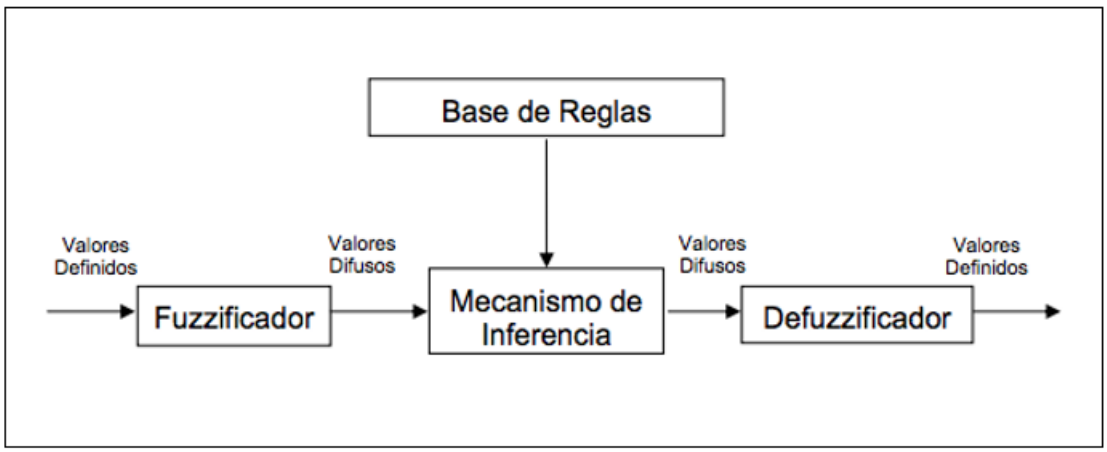


Figura. 1.1 Controlador Fuzzy

Recuperado de: Estructura básica de un Sistema Difuso (Ignacio & Domínguez, 2009)

1.4.3 Raspberry Pi 4

1.4.3.1 Introducción a la Raspberry 4

“La Raspberry Pi 4 Modelo B, es la primera de una nueva generación de computadoras Raspberry Pi que admiten más RAM y rendimiento de CPU, GPU y E/S significativamente mejorados.” (Raspberry Ltda, 2019)

1.4.3.2 Características y Capacidades de la Raspberry Pi 4

“La Raspberry Pi 4 tiene las siguientes características:

Hardware

- ARM-Cortex A72 de cuatro núcleos y 64 bits funcionando a 1,5 GHz.
- Opciones de RAM LPDDR4 de 1, 2 y 4 Gigabytes.
- Decodificación de hardware H.265 (HEVC) (hasta 4Kp60).
- Decodificación de hardware H.264 (hasta 1080p60).
- Gráficos 3D VideoCore VI
- Admite salida de pantalla HDMI dual de hasta 4Kp60.

Interfaces

- LAN inalámbrica 802.11 b/g/n/ac.
- Bluetooth 5.0 con BLE.
- 1 tarjeta SD.
- 2 puertos micro-HDMI que admiten pantallas duales con una resolución de hasta 4Kp60.
- 2 puertos USB2.
- 2 puertos USB3.
- 1 puerto Gigabit Ethernet (admite PoE con PoE HAT adicional).
- 1x puerto de cámara Raspberry Pi (MIPI CSI de 2 carriles).
- 1x puerto de pantalla Raspberry Pi (MIPI DSI de 2 carriles).
- GPIO de usuario 28x que admite varias opciones de interfaz:
    - Hasta 6x UART.
    - Hasta 6x I2C.
    - Hasta 5x SPI.
    - 1x interfaz SDIO.
    - 1x DPI (Pantalla RGB paralela).
    - 1x PCM.

- Hasta 2x canales PWM.
- Hasta 3x salidas GPCLK.

Software

- Conjunto de instrucciones ARMv8.
- Pila de software Linux madura.
- Desarrollado y mantenido activamente
- Soporte reciente del kernel de Linux.
- Área de usuario estable y con buen soporte.
- Disponibilidad de funciones de GPU utilizando API estándar.” (Raspberry Ltda, 2019)

1.4.3.3 Programación y Configuración de la Raspberry Pi 4

La Raspberry Pi 4 está disponible en modelos con capacidad de 2, 4 y 8 GB de RAM.

La lista de materiales necesarios para programar la placa de desarrollo es:

1- Laptop o PC
2- Raspberry Pi 4
3- Fuente de alimentación
4- Smartphone con función de Mobile Hotspot
5- Tarjeta Micro SD

Debemos tener sumo cuidado al momento de seleccionar la tarjeta Micro SD, pues en ella cargaremos el sistema operativo.

Es esencial tener en cuenta que la velocidad mínima recomendad para usar una Raspberry Pi es de 10 Mb/s, con una capacidad de la menos 8 Gb.

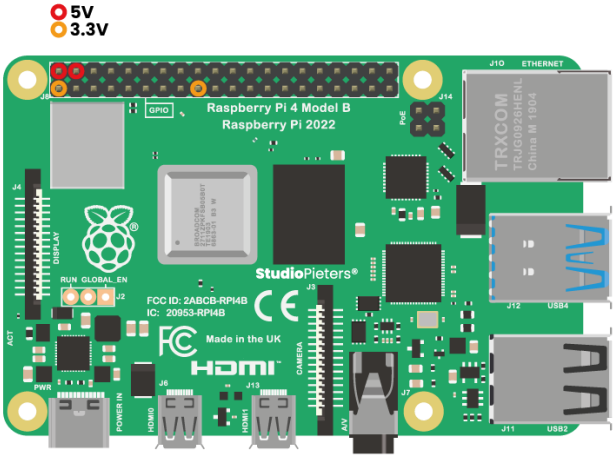


Figura. 1.2 ***Tarjeta Raspberry Pi 4***

Recuperado de: www.google.com

Como podemos observar en la figura 1.2, de manera similar a la mayoría de las computadoras de un solo usuario y microcontroladores, la Raspberry Pi tiene una entrada de alimentación nominal de 5V. No obstante, en la práctica, el voltaje puede tener pequeñas variaciones debido a las exigencias de la placa, y la mayoría de las fuentes USB suministran alrededor de +5,1 a +5,2V. Por consiguiente, al hacer cálculos aproximados para determinar la potencia (en vatios), comúnmente se considera un voltaje de una fuente USB de +5,15V. Esto se debe a que las fuentes de buena calidad tienden a esforzarse por mantener el voltaje alrededor de esta cifra, a pesar de las fluctuaciones rápidas en el consumo de corriente.

1.4.4 MATLAB y Simulink

MATLAB es un entorno de programación y software de cálculo numérico ampliamente empleado en ingeniería y ciencias aplicadas. Su sintaxis fácil de entender y sus potentes capacidades matemáticas lo convierten en una herramienta valiosa para una variedad de aplicaciones.

Principales Funciones de MATLAB en Ingeniería:

1. Cálculos Matriciales y Vectores: MATLAB destaca en operaciones con matrices y vectores, facilitando la manipulación y el análisis de datos.

2. Álgebra Lineal y Ecuaciones Diferenciales: Permite la resolución de sistemas de ecuaciones lineales, ecuaciones diferenciales y operaciones algebraicas avanzadas.

3. Visualización de Datos: Proporciona herramientas gráficas para visualizar datos, simplificando la interpretación de resultados.
4. Procesamiento de Señales e Imágenes: MATLAB se utiliza en el procesamiento de señales, filtrado y análisis de imágenes, siendo aplicable en áreas como la visión por computadora.
5. Desarrollo de Algoritmos y Modelado: Ingenieros utilizan MATLAB para desarrollar y probar algoritmos, así como para crear modelos matemáticos y simular sistemas.

1.4.4.1 Modelado y Simulación de Sistemas con Simulink

Simulink es una herramienta de MATLAB que facilita el modelado, simulación y análisis de sistemas dinámicos. Utiliza un enfoque gráfico basado en bloques, lo que simplifica la representación visual de sistemas complejos.

La facilidad que ofrece Simulink al realizar la representación de los sistemas mediante bloques gráficos que visualizan componentes y conexiones nos resulta bastante útil en el modelado de la planta de control, permitiéndonos simular el comportamiento dinámico del sistema en tiempo real y simplificando la evaluación de los diseños previos a la implementación.

Al analizar la respuesta del sistema podemos ver cómo responden a las diversas entradas y condiciones iniciales.

Una gran ventaja que tenemos al usar este entorno es a facilidad de crear de manera automática el código a partir de modelo de Simulink para su aplicación en el sistema embebido.

1.4.4.2 Integración de MATLAB y Simulink en Proyectos de Control

Al combinar estos dos entornos, podemos sacar el mayor provecho posible al momento de realizar el modelado de la planta y obtener el modelo matemático.

Las tareas clave que podemos realizar al combinar los entornos son las siguientes:

- Diseño de controladores
- Simulación de sistemas dinámicos
- Análisis de estabilidad
- Generación de código para controladores

Utilizando MATLAB, es posible diseñar y ajustar controladores, analizando el rendimiento del sistema en tiempo real y al utilizar Simulink nos facilita la simulación de sistemas dinámicos, lo que nos permite evaluar el comportamiento del sistema antes de la implementación real.

Por otro lado, se pueden realizar análisis detallados sobre la estabilidad y la respuesta transitoria de sistemas de control utilizando las herramientas proporcionadas por MATLAB y Simulink.

Y finalmente, la combinación de MATLAB y Simulink nos permite generar código para la implementación de controladores en hardware embebido, facilitando la transición del diseño a la implementación práctica.

1.4.5. Internet Industrial de las Cosas (IIoT)

El IIOT se refiere al uso de la tecnología de Internet de las Cosas en entornos Industriales, permitiendo así el análisis más profundo y aprovechamiento de los datos generados por los dispositivos IIoT con la finalidad de mejorar la eficiencia, productividad y visibilidad.

Las redes IIoT admiten la comunicación máquina a máquina y la transmisión regular de los datos entre el sistema central y todos los dispositivos integrados en IIoT.

1.4.5.1 Concepto y Cómo funciona el Internet Industrial de las Cosas:

“El Internet Industrial de las Cosas (IIoT) es un subgrupo del Internet de las Cosas (IoT), la diferencia entre las dos es que mientras el IoT está enfocado a servicios para los consumidores, el IIoT se concentra en aumentar la seguridad y la eficiencia en las aplicaciones Industriales.’’ (Sisinni, Saifullah, Han, & Gidlund., 2018)

El IIoT consta de los siguientes factores:

1. Dispositivos conectados capaces de detectar, comunicar y almacenar información sobre sí mismos.
2. Red de comunicación de datos, ya sea pública o privada.
3. Evaluación y programas que derivan información empresarial a partir de datos en estado bruto.
4. Almacenamiento de la información producida por los dispositivos IoT.

1.4.5.2 Conexión a la Nube y Recolección de Datos en Proyectos IIoT

Figura. 1.3 Ilustración de enlace entre la nube y la industria

Recuperado de: La Internet de las cosas

Los datos que recopilamos mediante dispositivos IIoT son transmitidos a la nube para su almacenamiento, procesamiento y análisis. Al realizar esta práctica damos paso al acceso a los datos desde cualquier ubicación y simplifica la implementación de servicios basados en la nube.

Una de las ventajas de tener una conexión a la nube es obtener una capacidad de almacenamiento escalable para manejar grandes volúmenes de datos que generan los dispositivos IIoT. Posteriormente, esos datos pueden procesarse de manera centralizada permitiendo el análisis correspondiente y la generación de información.

1.4.5.3 Beneficios del IIoT en Sistemas de Control

Gracias a la implementación del Internet de las Cosas Industrial podemos tener una mejora de procesos, debido a que los sensores IIoT posibilitan la recopilación de datos en tiempo real, ofreciendo una información más precisa y actualizada de las variables del sistema.

Todos los datos recopilados pueden emplearse para realizar análisis predictivos, anticipando posibles fallos o necesidades de mantenimiento a la planta.

Otra de las ventajas de utilizar IIoT en sistemas de control es el control preciso de los procesos lo que reduce el consumo de energía, optimizando recursos.

Finalmente, incorporar la conectividad IIoT nos da la ventaja de supervisar y controlar el sistema desde ubicaciones remotas, mejorando la eficiencia operativa.

CAPÍTULO 2

2. METODOLOGÍA

Para realizar la metodología de nuestro proyecto se usó el esquema de trabajo que se muestra en la Figura 2.1, en la cual se observan paso a paso las acciones realizadas, empezando por la selección de materiales y escatimar costos, para luego realizar el diseño mecánico de la planta utilizando el programa Inventor. Posterior a eso se realizó el diseño eléctrico de la planta utilizando el programa Fritzing. Finalmente se realizó el prototipo a escala de la planta de fluidos conectando todos los elementos y proceder con las pruebas.

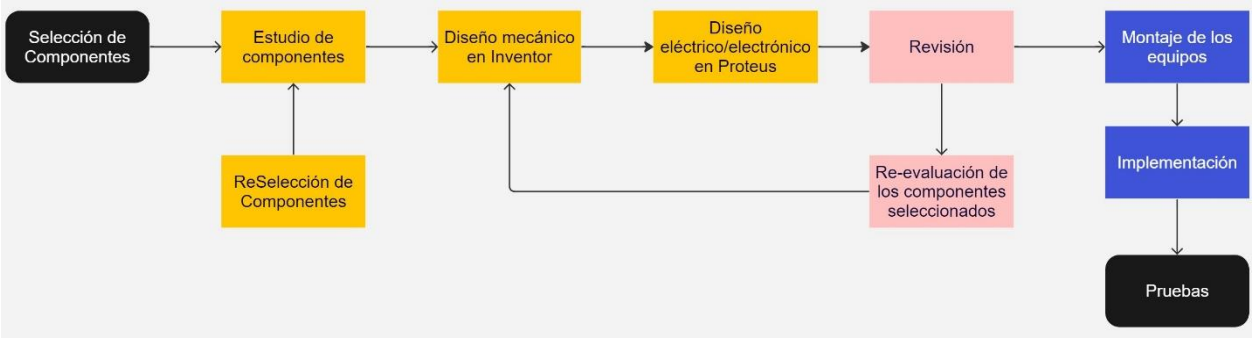


Figura. 2.1 Diagrama de flujo para la solución

Autores: Saul Carvallo & Jonathan Zambrano

2.1 Selección de Componentes

Se seleccionó la tarjeta Raspberry Pi 4 Modelo B debido a que sus características se adecuaban perfectamente a nuestro proyecto, tiene los pines con funciones específicas adaptadas a lo que se requiere hacer.

Para poder visualizar el comportamiento de nuestra planta fue necesario conseguir una pantalla para Raspberry Pi de 7 pulgadas. Los demás componentes se muestran en la Tabla 2.1.

LISTA DE COMPONENTES	
Material	**Precio**
Raspberry pi 4 Modelo B de 8Gb	$150,00
Impresión 3D para el circuito	$30,00
Balde de 12 L de capacidad	$8,00
Balde de 16 L de capacidad	$10,00
Pantalla Raspberry Pi de 7 pulgadas IPS 1024 x 600 HDMI	$75,00
Bomba de agua (12v, 2A, 3.5l/min, 0.48MPa)	$35,00
Módulo IRF520	$2,50
Electroválvula de 12V Plástica	$11,00
Módulo Relé 5V 2 Canales	$4,00
Fuente 12V - 10ª	$15,00
Cargador Raspberry 5V - 3ª	$10,00
Case Acrílico + Ventilador	$8,00
Memoria Micro SD Tipo 10 de 16Gb	$6,50
TOTAL	$365,00

Tabla 2.1: Listado de componentes

Autores: Saul Carvallo & Jonathan Zambrano

2.2 Diseño mecánico de la planta.

Dentro del diseño mecánico se deben tomar varias consideraciones puesto que es el que valida la estructura que sostendrá todo el sistema embebido, algunas de estas son Integración, ensamblaje, diseño ergonómico, facilidad en manufactura. en resumen, todas estas apreciaciones dentro del diseño mecánico de en prototipos electrónicos es esencial para brindar seguridad dentro de parámetros de funcionalidad, protección y uso, además de la viabilidad de fabricación de estos dispositivos.

2.2.1 Modelo de la plataforma de la planta

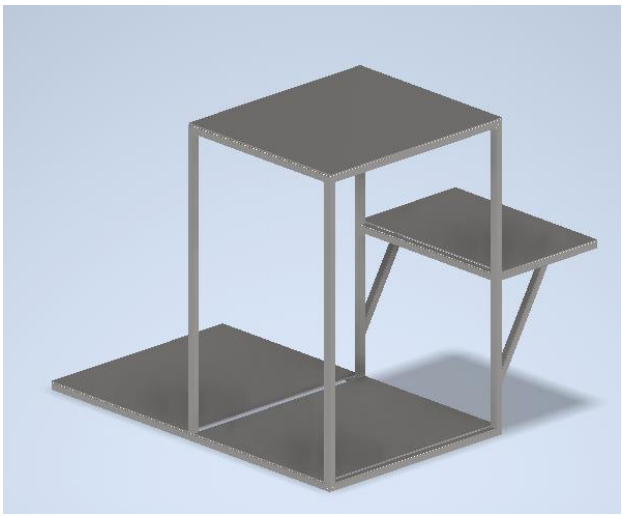


Figura. 2.2 Modelo de la plataforma de la planta embebida.

Autores: Saul Carvallo & Jonathan Zambrano

Dentro de la plataforma diseñada se tiene una estructura de tubo hueco de 1.2mm de espesor y un diámetro de 1”, en donde se tiene la consideración acorde al peso de cada nivel, teniendo dos tanques, uno de 16 litros en el nivel superior y uno de 12 litros en el nivel inferior, cabe destacar en el diseño puntales a 45 grados que dan soporte al nivel donde va el tablero con la fuente y el sistema de control general.

2.2.2 Modelo del montaje de los elementos de la planta

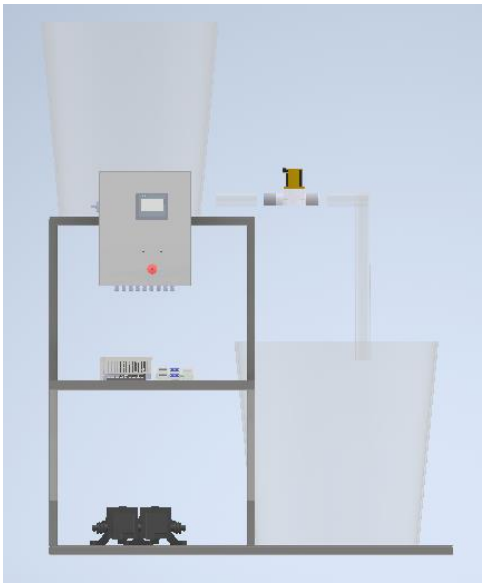


Figura. 2.3 Vista representativa con los elementos electrónicos montados en campo.

Autores: Saul Carvallo & Jonathan Zambrano

2.3 Dimensionamiento de los componentes y equipos

Para el dimensionamiento general de la planta donde se incluyen todos los sistemas electrónicos embebidos se usa el método de diseño basado en especificaciones técnicas en donde el alcance electrónico parte en base a la capacidad de los tanques puesto que de aquí se pude obtener la capacidad de la bomba para hacer el llenado en el menor tiempo posible, además de esto se obtiene la capacidad de la fuente para evitar caídas de voltaje y que el sistema opere en los rangos de diseño permitidos que son entre 1-9 litros.

2.3.1 Dimensionamiento mecánico

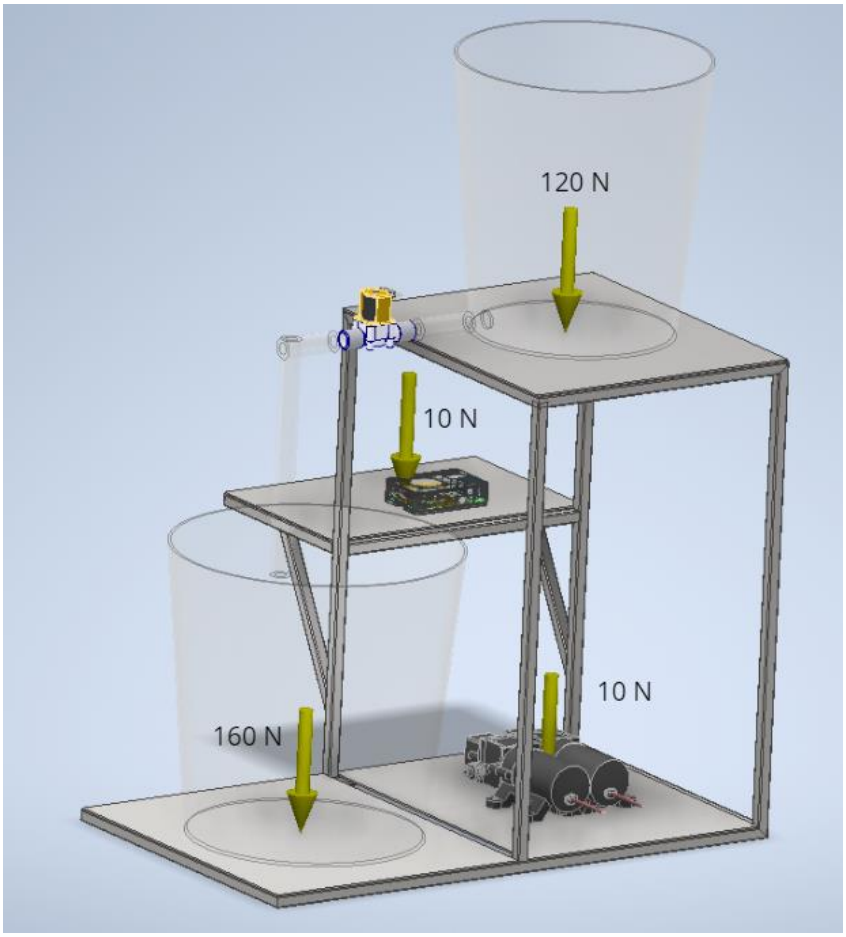


Figura. 2.4 Fuerza aplicada en cada sección de la estructura mecánica.

Autores: Saul Carvallo & Jonathan Zambrano

Considerando la figura 2.4 en donde se tienen las fuerzas en cada sección para poder dar soporte al montaje de los equipos en la estructura metálica, se realizó el cálculo del factor de seguridad de la plataforma creada en Inventor.

El factor de seguridad adecuado siempre debe ser mayor a 1, caso contrario la estructura será considerada no apta, puesto que se estaría manejando un umbral en niveles bajos con lo cual comprometería todo el sistema de la estructura mecánica detallada a continuación.

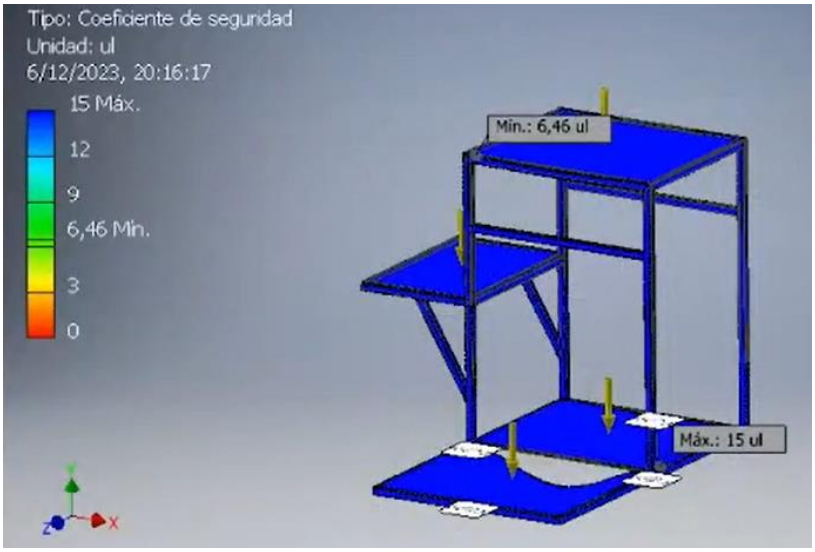


Figura. 2.5 Factor de seguridad con umbral mínimo de 6.46 y máximo de 15.

Autores: Saul Carvallo & Jonathan Zambrano

En vista a lo establecido anteriormente en la figura 2.5 podemos observar que nuestros índices de factor de seguridad se encuentran dentro de los parámetros establecidos con lo cual se puede validar el diseño mecánico presentado en esta sección, por lo que ese da por finalizado esta etapa y proceder al montaje en campo de los dispositivos electrónicos y de control que gobiernan a la planta.

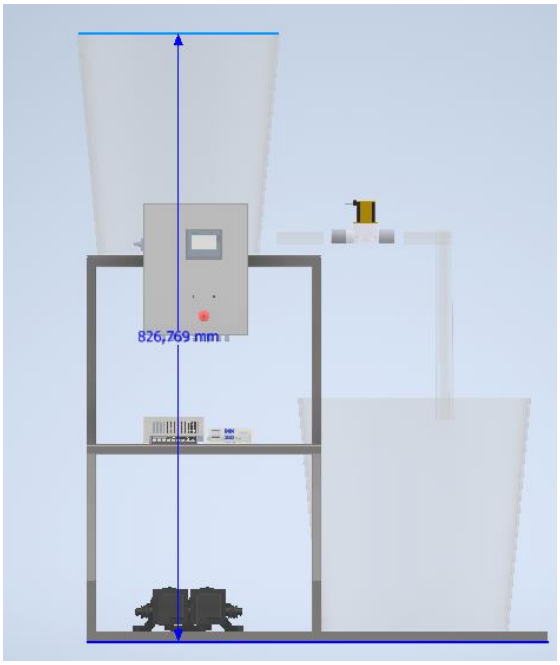


Figura. 2.6 Altura del prototipo considerando tanque.

Autores: Saul Carvallo & Jonathan Zambrano

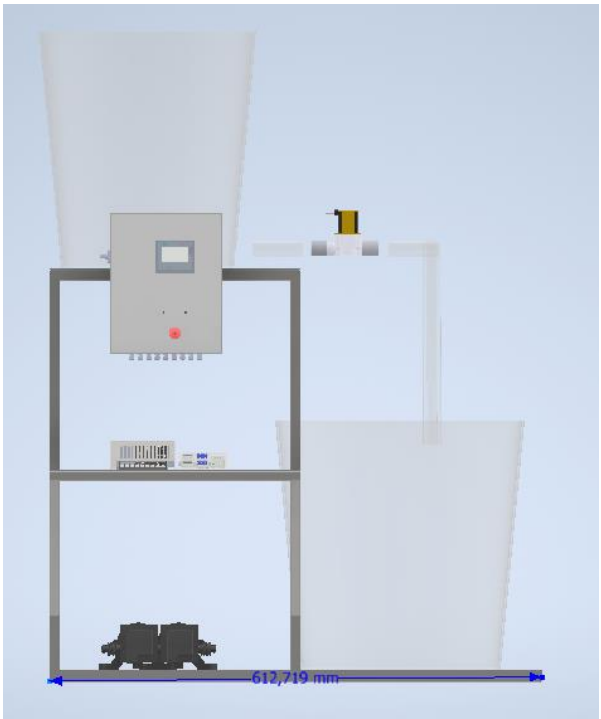


Figura. 2.7 Ancho del prototipo final.

Autores: Saul Carvallo & Jonathan Zambrano

2.3.2 Diseño eléctrico-electrónico en proteus

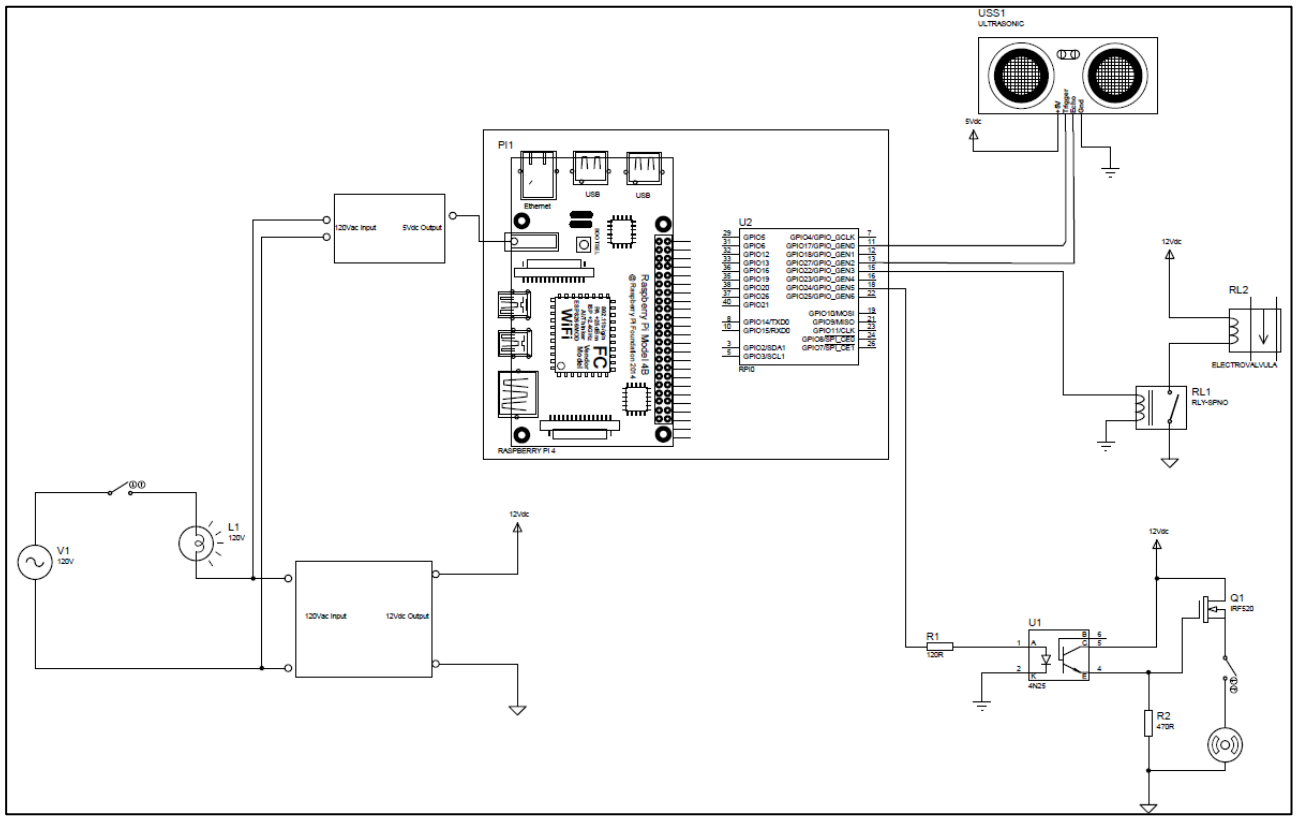


Figura. 2.8 Diagrama general de conexiones, eléctricas y electrónicas.

Autores: Saul Carvallo & Jonathan Zambrano

2.3.3 Dimensionamiento eléctrico

2.3.3.1 Fuente

Se utilizará una fuente que trabaja a 12 V – 10 A, con la cual se energizará al sistema. Los datos técnicos se observan en la Tabla 2.2.

Figura. 2.9 Fuente 12VDC

Autores: Saul Carvallo & Jonathan Zambrano

Datos técnicos de la fuente	
Voltaje de Salida	12 VDC
Corriente Máxima	10 A
Potencia Nominal	120 W
Rizado y Ruido	120 mV P-P
Tiempo de activación	20 ms/230VAC
Voltaje de entrada	AC 100 – 120 V
Eficiencia	>80%

Tabla 2.2 Datos técnicos de la fuente

Autores: Saul Carvallo & Jonathan Zambrano

2.3.3.2 Raspberry Pi

El controlador que se usará es la Raspberry Pi 4 modelo B, la cual estará encargada de toda la parte del control de proceso. Sus datos técnicos los pudimos observar en el punto 1.4.3.2.

2.3.3.3 Pantalla Raspberry Pi de 7 pulgadas IPS 1024 x 600 HDMI

Para observar la interfaz hombre-máquina, usaremos la pantalla de 7 pulgadas para Raspberry Pi, que puede ser manejada mediante su pantalla táctil. Sus datos técnicos los observamos en la tabla 2.3.

Figura. 2.10 Pantalla LCD Raspberry Pi

Autores: Saul Carvallo & Jonathan Zambrano

Datos técnicos de la pantalla	
Tamaño de pantalla	7 pulgadas
Resolución máxima de pantalla	1024 x 600 px
Marca	SunFounder
Características especiales	Pantalla táctil
Tecnología de conectividad	USB, HDMI

Tabla 2.3 Datos técnicos de la pantalla raspberry pi de 7 pulgadas

Autores: Saul Carvallo & Jonathan Zambrano

2.3.3.4 Bomba de Agua

La bomba a utilizar tiene un diseño optimizado que logra un equilibrio preciso entre el consumo energético, el caudal y la presión, resaltando su importancia en aplicaciones que demandan una gestión eficiente de los recursos hídricos. Este estudio proporcionará un análisis detallado de sus características técnicas y su aporte a la gestión sostenible del agua. Sus características las observamos en la tabla 2.4.

Figura. 2.11 Bomba de agua

Autores: Saul Carvallo & Jonathan Zambrano

Bomba de Agua (12v, 2A, 7-9L/min, 0.85MPa)	
Voltaje	12 V
Corriente	2A
Presión	0.85 MPa
Flujo	7-9 l/min
Peso	0.6 Kg
Entrada y salida de agua	Puerto de tornillo único de 181,80 m

Tabla 2.4 Datos técnicos de la bomba de agua

Autores: Saul Carvallo & Jonathan Zambrano

2.3.3.5 Relé SRD – 5VDC – SL – C

Al utilizar este relé aprovechamos su habilidad de supervisar el circuito a través de una señal de bajo voltaje. Sus características las podemos observar en la tabla 2.5.

Figura. 2.12 Relé 5VDC

Autores: Saul Carvallo & Jonathan Zambrano

Relé SRD – 5VDC – SL – C	
Voltaje de operación bobina	5VDC
Corriente bobina	75 mA
Voltaje máximo de carga	240V AC / 30 V DC
Corriente máxima de carga	10A
Contactos	1 NA, 1 NC
Tiempo de acción	10 ms / 5 ms

Tabla 2.5 Datos técnicos del relé

Autores: Saul Carvallo & Jonathan Zambrano

2.3.3.6 Sensor ultrasónico HC-SR04

Con este sensor sacaremos la lectura de la distancia para obtener el volumen del recipiente que en este caso es un cilindro, teniendo el valor de la altura de este para aplicar la fórmula $V = \pi r^2 h$

Al obtener el valor de estas variables ya tendremos el valor del nivel del tanque. Las características del sensor las podemos ver en la tabla 2.6.

Figura. 2.13 Sensor ultrasónico

Autores: Saul Carvallo & Jonathan Zambrano

Sensor ultrasónico HC-SR04	
Distancia	2 cm – 450 cm
Voltaje de Operación	5 VDC
Duración mínima del pulso de disparo	10 µs
Duración del pulso eco de salida	100 – 25000 µs
Tiempo mínimo de espera entre medidas	20 ms
Frecuencia de ultrasonido	40 KHz

Tabla 2.6 Datos técnicos del sensor ultrasónico

Autores: Saul Carvallo & Jonathan Zambrano

2.3.3.7 Electroválvula

Nos permite dar paso o cerrar el flujo de agua en el circuito. Funciona de forma on/off, es decir que cuando el nivel sea menor del que le pedimos en los parámetros, la electroválvula va a dejar salir el agua y llenar el tanque hasta el punto deseado. Sus características las podemos observar en la tabla 2.7.

Figura. 2.14 Electroválvula de uso industrial

Autores: Saul Carvallo & Jonathan Zambrano

Electroválvula	
Voltaje de operación	12 VDC
Corriente de operación	0.6 A
Potencia de consumo	8 W
Temperatura de funcionamiento	5°C a 100°C
Presión de funcionamiento mínima	0.02 MPa
Presión de funcionamiento máxima	0.8 MPa
Tiempo de respuesta	≤ 0.15 s

Tabla 2.7 Datos técnicos de la electroválvula

Autores: Saul Carvallo & Jonathan Zambrano

2.3.4 Montaje de los equipos

Figura. 2.15 Prototipo de la planta implementado en diseño mecánico validado.

Autores: Saul Carvallo & Jonathan Zambrano

Como podemos observar en la Figura 2.15, se realizó el montaje del tablero donde están las conexiones electrónicas que darán funcionamiento al proyecto, así como los demás materiales de la Tabla 1.1.

Debido a que el factor de seguridad que mencionamos en el punto 2.3.1. implementado fue el adecuado, la estructura soporta correctamente todos los elementos encima de ella, permitiendo así el funcionamiento de la planta sin presentar inconvenientes.

2.3.5 Diagrama de flujo del controlador PID

Como se puede apreciar en la figura 2.16 tenemos el diagrama de flujo funcional para el controlador PID, en donde tenemos que después de una sintonización tenemos los valores de las constantes Kp,Ki,Kd ,luego procedemos a ingresar el setpoint de nivel del sistema y se incia el bucle de lectura del sensor de nivel(ultrasonico) y la codificación para accionar nuestro actuador que en este caso es la bomba.

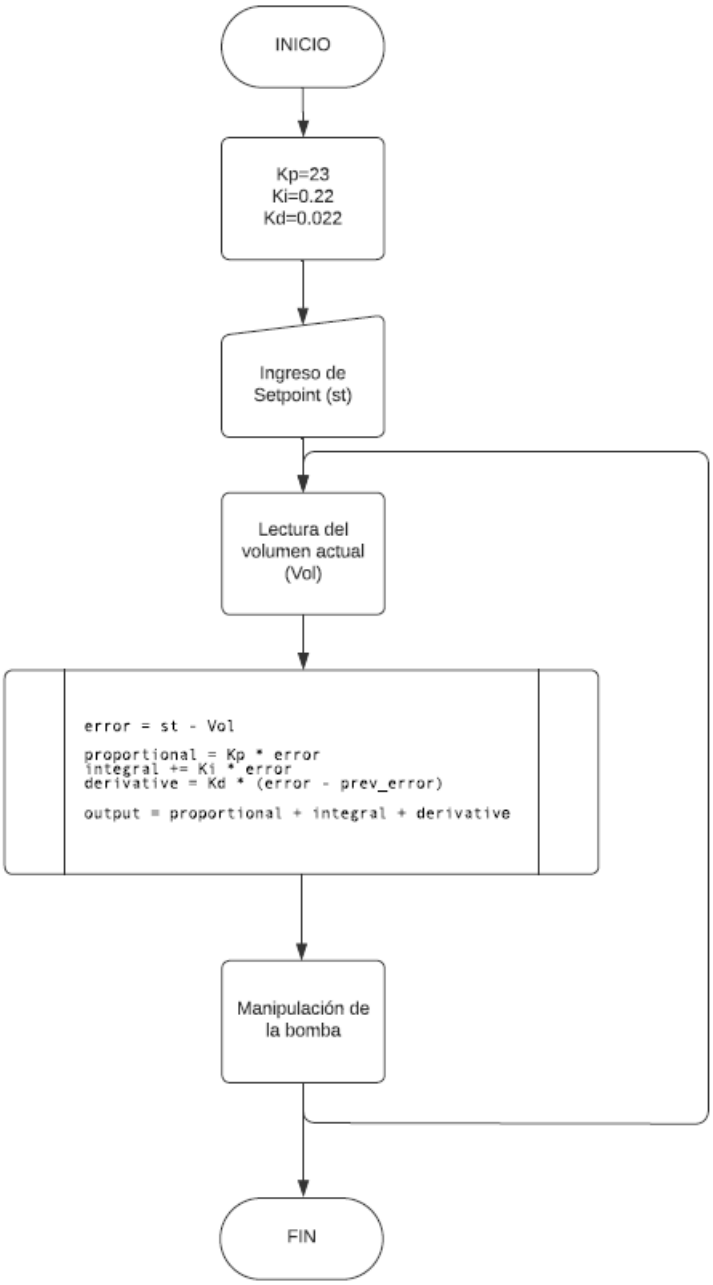


Figura. 2.16 Diagrama de flujo del controlador PID del prototipo

Autores: Saul Carvallo & Jonathan Zambrano

2.3.5.1 Diagrama de flujo del controlador Fuzzy

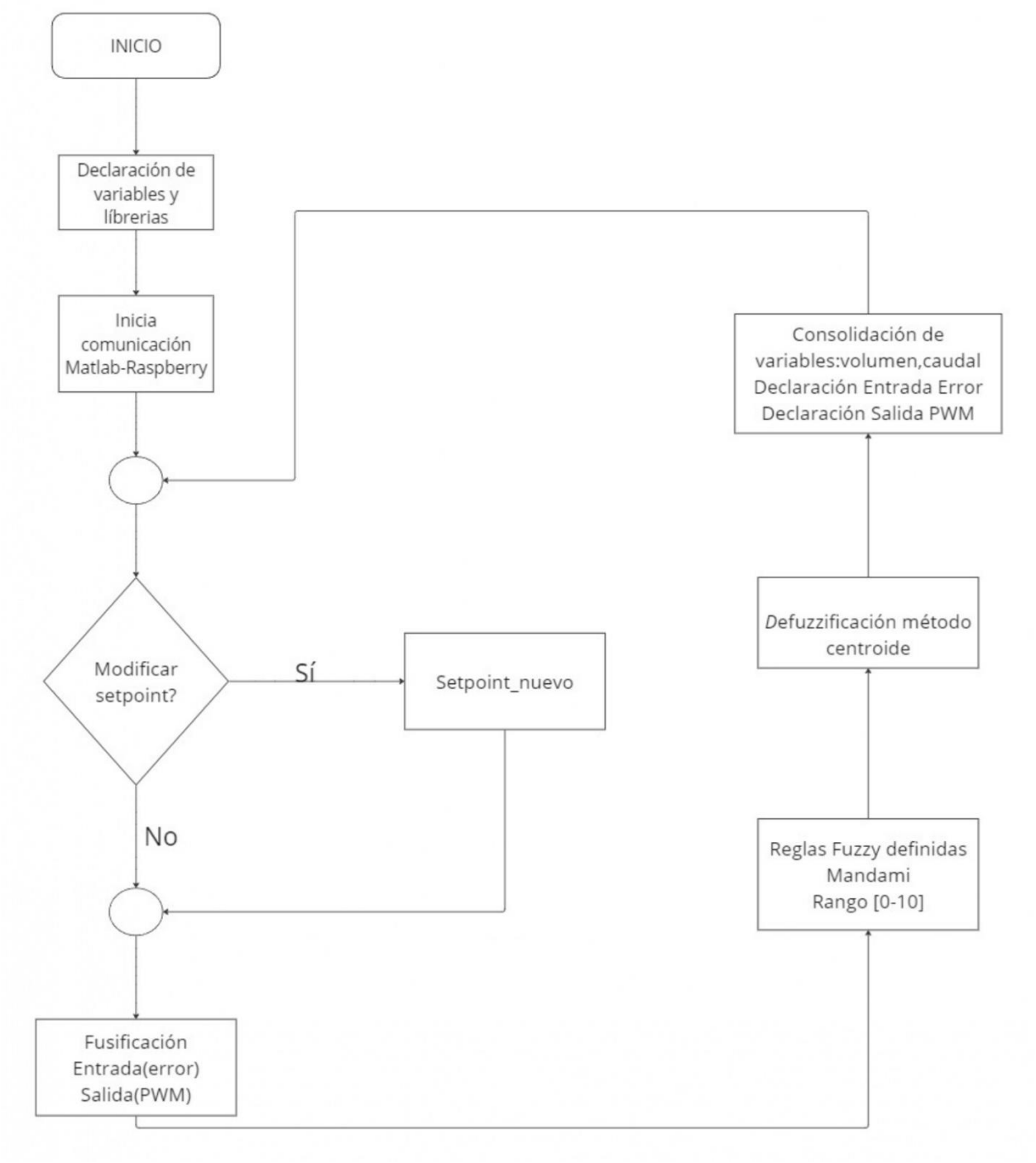


Figura 2.17 Diagrama de flujo del controlador Fuzzy del prototipo

Autores: Saul Carvallo & Jonathan Zambrano

2.3.5.2 Reglas del controlador Fuzzy

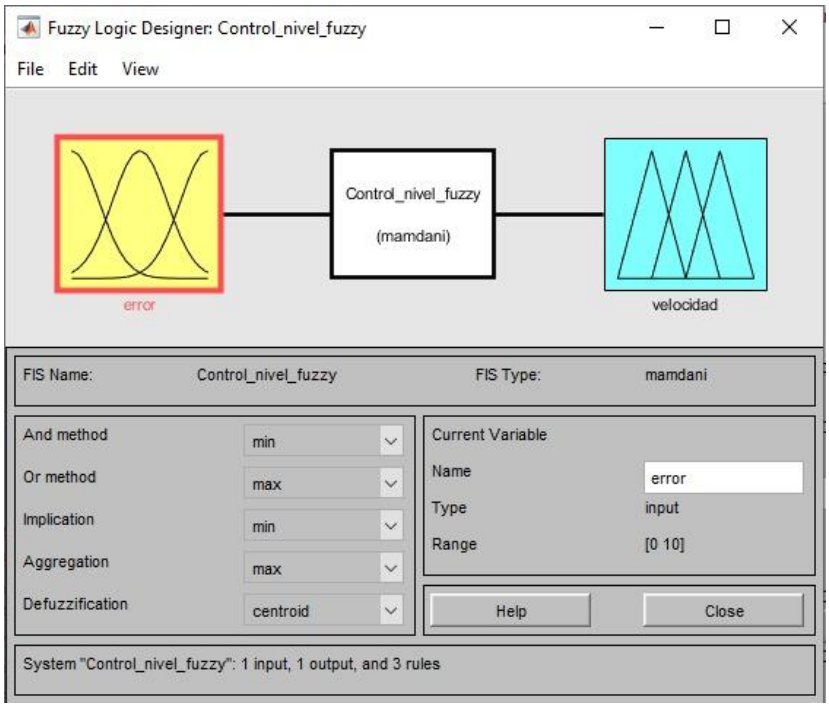


Figura 2.18 Reglas del controlador Fuzzy

Autores: Saul Carvallo & Jonathan Zambrano

2.3.5.3 Ingreso a la raspberry

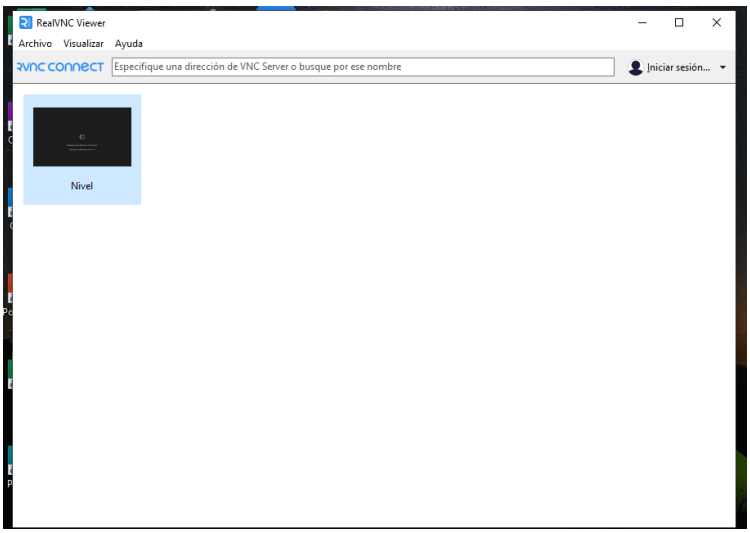


Figura. 2.19 Interfaz de inicio de la Raspberry.

Autores: Saul Carvallo & Jonathan Zambrano

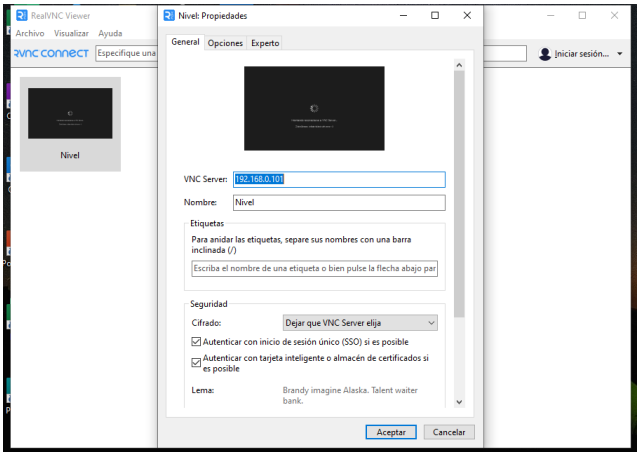


Figura. 2.20 Configuración del servidor a utilizar.

Autores: Saul Carvallo & Jonathan Zambrano

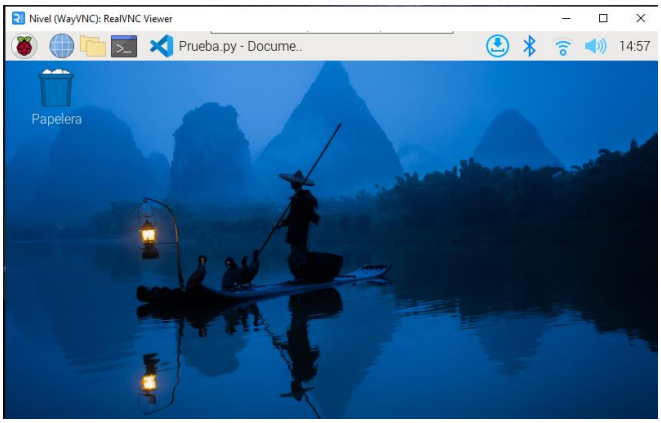


Figura. 2.21 Pantalla de inicio de la Raspberry.

Autores: Saul Carvallo & Jonathan Zambrano

2.3.5.4 Programación realizada en visual studio code dentro de la raspberry

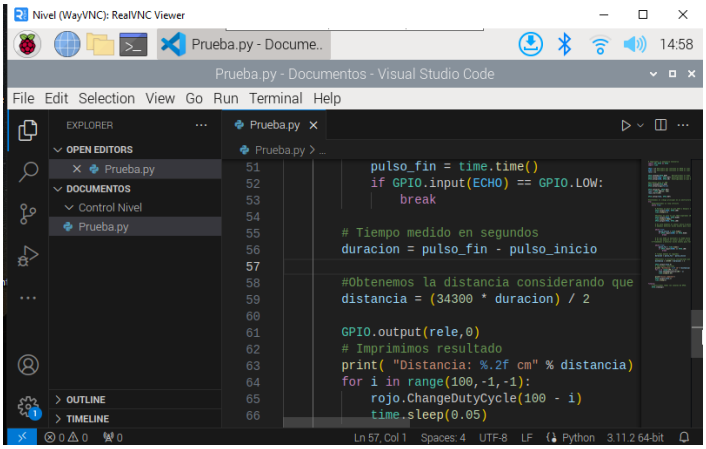


Figura. 2.22 Visual Studio Code embebido dentro de la Raspberry.

Autores: Saul Carvallo & Jonathan Zambrano

2.3.5.5 Código para el control de nivel

El código mostrado en el punto 1.1 de los anexos, nos permite realizar mediciones de distancia utilizando un sensor ultrasónico (HC-SR04) conectado a una Raspberry Pi. A continuación, se presenta una explicación detallada del código:

1. Configuración de Pines y Control PWM:

- Se asignan las variables ‘TRIG’, ‘ECHO’, y ‘rele’ para representar los pines GPIO utilizados.
- Los pines ‘TRIG’ y ‘ECHO’ se configuran como salida y entrada, respectivamente.
- Se inicializa el sistema de pines GPIO y se utiliza la numeración BCM para la Raspberry Pi.
- Además, se configura un pin de la Raspberry Pi (‘24’) para el control de un LED mediante PWM (‘rojo’), comenzando con un ciclo de trabajo del 100% (máxima luminosidad del LED).

2. Bucle Principal:

- Se implementa un bucle infinito para llevar a cabo mediciones continuas de distancia.
- Se envía un pulso ultrasónico (‘TRIG’ en alto) y se espera un breve período para permitir que el transductor se estabilice (‘time.sleep(0.5)’).
- La medición del tiempo se inicia cuando el sensor envía pulsos ultrasónicos (‘ECHO’ en alto).
- Se calcula la duración del pulso ultrasónico recibido (‘duracion’) y se utiliza para determinar la distancia al objeto mediante la velocidad del sonido.
- La distancia medida en centímetros se imprime en la consola.
- Se utiliza un bucle para variar gradualmente el ciclo de trabajo del LED (‘rojo’), creando un efecto visual de cambio de intensidad luminosa.
- Se controla un relé (‘rele’), alternando entre encendido y apagado, y se espera 2 segundos antes de realizar la siguiente medición.

3. Manejo de Errores y Limpieza:

- Todo el código está encapsulado en bloques ‘try’ y ‘finally’ para manejar posibles errores y asegurar que se realice una limpieza adecuada de los pines GPIO al salir del programa.

- En el bloque ‘finally’, se emplea ‘GPIO.cleanup()’ para restablecer todos los canales de GPIO a su estado original.

2.3.5.6 Código para la lectura en la raspberry

Editor - C:\Users\saulc\Desktop\Interface_Matlab\Lectura_raspberry.m

Enviar_IOT.m | Control_nivel_fuzzy.fis | Seleccionado controlador FUZZY.ps | HMI.m | Lectura_raspberry.m

```
clear all
clc

% Conectar con la Raspberry Pi
mypi = raspi();

% Ejecutar el script de Python y leer la salida
disp("Inicializacion")
command = 'python3 /home/pi/Documents/Lectura/lectura_sensor.py';
disp("Comando")

for i=0:10
    distancia = system(mypi, command);
    disp("lectura de distancia")

    % Procesar y mostrar la distancia
    disp(['La distancia medida es: ', num2str(distancia), ' cm']);
end
```

Figura. 2.23 Código para la lectura en la raspberry

Autores: Saul Carvallo & Jonathan Zambrano

Este código en MATLAB realiza varias operaciones relacionadas con la Raspberry Pi y el sensor de distancia, las cuales tenemos:

1. Limpiar el Entorno:
- ‘clear all’: Elimina todas las variables existentes en el espacio de trabajo.
- ‘clc’: Borra el contenido de la ventana de comandos.

2. Conexión con la Raspberry Pi:
- ‘mypi = raspi()’: Establece una conexión con la Raspberry Pi y asigna el objeto resultante a ‘mypi’.

3. Ejecutar el Script de Python:
- ‘command’: Almacena el comando de sistema para ejecutar el script de Python ‘lectura_sensor.py’ ubicado en ‘/home/pi/Documents/Lectura/’.
- ‘for i=0:10’: Inicia un bucle que realiza la operación siguiente 11 veces (índices de 0 a 10).
- ‘distancia = system(mypi, command)’: Ejecuta el comando de sistema en la Raspberry Pi utilizando el objeto ‘mypi’ y obtiene la salida del script Python, que representa la distancia medida por el sensor.

4. Procesar y Mostrar la Distancia:
- ‘disp(['La distancia medida es: ', num2str(distancia), ' cm'])’: Muestra en la consola de MATLAB la distancia medida en centímetros en cada iteración del bucle.

CAPÍTULO 3

3 RESULTADOS Y ANÁLISIS

3.1 Fabricación de la estructura metálica y montaje de los equipos

En la Figura 3.1 mostrada se observa las bases para la estructura metálica hechas de metal galvanizado, que van a sostener tanto el tablero de conexiones como los baldes de 12 y 16 litros respectivamente.

Figura. 3.1 Planchas de acero galvanizado para el prototipo base.

Autores: Saul Carvallo & Jonathan Zambrano

Figura. 3.2 Tubos huecos con diámetro de 1 pulgada

Autores: Saul Carvallo & Jonathan Zambrano

En la Figura 3.2 observamos el proceso de soldadura de los tubos huecos con diámetro de 1”, con un espesor de 1.2 mm, dando como resultado final la imagen que observamos en la Figura 3.3.

Figura. 3.3 Estructura metálica finalizada

Autores: Saul Carvallo & Jonathan Zambrano

Posterior a esto, realizamos las pruebas de montaje de los elementos comprobando que el factor de seguridad que mencionamos en el punto 2.3.1. fue el adecuado, lo cual podemos observar en la Figura 3.4.

Figura. 3.4 Pruebas de montaje de los elementos

Autores: Saul Carvallo & Jonathan Zambrano

Para un mejor acabado se realizó una aplicación de capa de pintura como podemos observar en la Figura 3.5.

Figura. 3.5 Aplicación de capa de pintura a la estructura metálica

Autores: Saul Carvallo & Jonathan Zambrano

Dado todo esto, se procedió a realizar el montaje de los elementos dando como resultado un prototipo a escala de planta de fluidos, el cual podemos observar en las Figuras 3.6, 3.7 y 3.8.

Figura. 3.6 Montaje de los elementos de la planta de fluidos

Autores: Saul Carvallo & Jonathan Zambrano

Figura. 3.7 Montaje de los elementos de la planta de fluidos

Autores: Saul Carvallo & Jonathan Zambrano

Figura. 3.8 Montaje de los elementos de la planta de fluidos

Autores: Saul Carvallo & Jonathan Zambrano

3.2 Modelo de la función de transferencia en Matlab

04/01/24 19:42 C:\Users\jo...\Funcion Transferencia.m 1 of 1

```
%% Autores :Jonathan Zambrano/Saul Carvallo
%% Materia integradora
% Datos proporcionados (tiempo en segundos y litros de llenado)
tiempo = 0:1:11; % Supongamos un intervalo de muestreo de 1 segundo
litros = [1.81, 2.5, 3, 3.5, 4, 4.8, 5.45, 5.94, 6.53, 7.12, 7.71, 8.3];

% Crear un objeto de datos de identificación
data = iddata(litros', ones(size(litros')), 1, 'SamplingInstants', tiempo);

% Estimar la función de transferencia
modelo_identificado = tfest(data, 1, 1)

% Mostrar la función de transferencia identificada
disp('Función de Transferencia Identificada:');
disp(modelo_identificado);

% Crear un vector de tiempo para la simulación
tiempo_simulacion = linspace(min(tiempo), max(tiempo), 100);

% Simular el modelo identificado
litros_simulados = lsim(modelo_identificado, ones(size(tiempo_simulacion)),↙
tiempo_simulacion);

% Graficar los datos y la simulación del modelo identificado
figure;
plot(tiempo, litros, 'o', 'DisplayName', 'Datos experimentales');
hold on;
plot(tiempo_simulacion, litros_simulados, 'r', 'DisplayName', 'Modelo↙
identificado');
xlabel('Tiempo (s)');
ylabel('Litros de llenado');
legend;
grid on;
```

3.2.1 Función de transferencia

```
modelo_identificado =

  From input "u1" to output "y1":
  1.773 s + 0.5727
  ----------------
   s + 6.103e-11

Continuous-time identified transfer function.

Parameterization:
   Number of poles: 1   Number of zeros: 1
   Number of free coefficients: 3
   Use "tfdata", "getpvec", "getcov" for parameters and their uncertainties.
```

Figura. 3.9 Modelo de identificación de la planta prototipada.

Autores: Saul Carvallo & Jonathan Zambrano

3.2.1.1 Grafica de función de transferencia

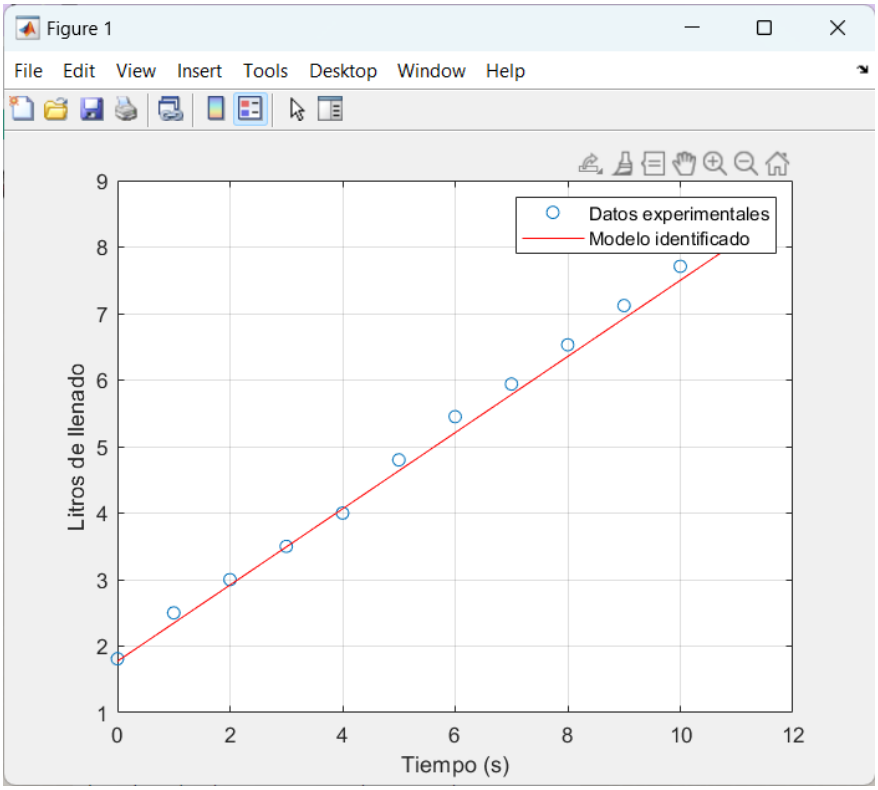


Figura. 3.10 Grafica comparativa de datos experimentales y modelo identificado

Autores: Saul Carvallo & Jonathan Zambrano

3.2.2 Sintonización del controlador

Para la sintonización del controlador PID después de realizar varias pruebas, se optó por utilizar el valor de 50 para la variable proporcional Kp, el valor 0.4 para la variable derivativa Kd, y 0.01 para la variable integral Ki, siendo éstos los valores promedio a usar para un rango de volumen que va de 2 litros hasta 5.8 litros. El error promedio resultó ser 0.375092, esto nos da un margen de diferencia entre el volumen medido con el volumen real de 0.375092.

Para la sintonización del controlador Fuzzy después de realizar varias pruebas, se optó por utilizar el valor de 50 para la variable proporcional Kp, el valor 0.4 para la variable derivativa Kd, y 0.022 para la variable integral Ki, siendo éstos los valores promedio a usar para un rango de volumen que va de 2 litros hasta 5.8 litros. El error promedio resultó ser 0.326871 con un PWM promedio de 0.460274, esto nos da un margen de diferencia entre el volumen medido con el volumen real de 0.326871.

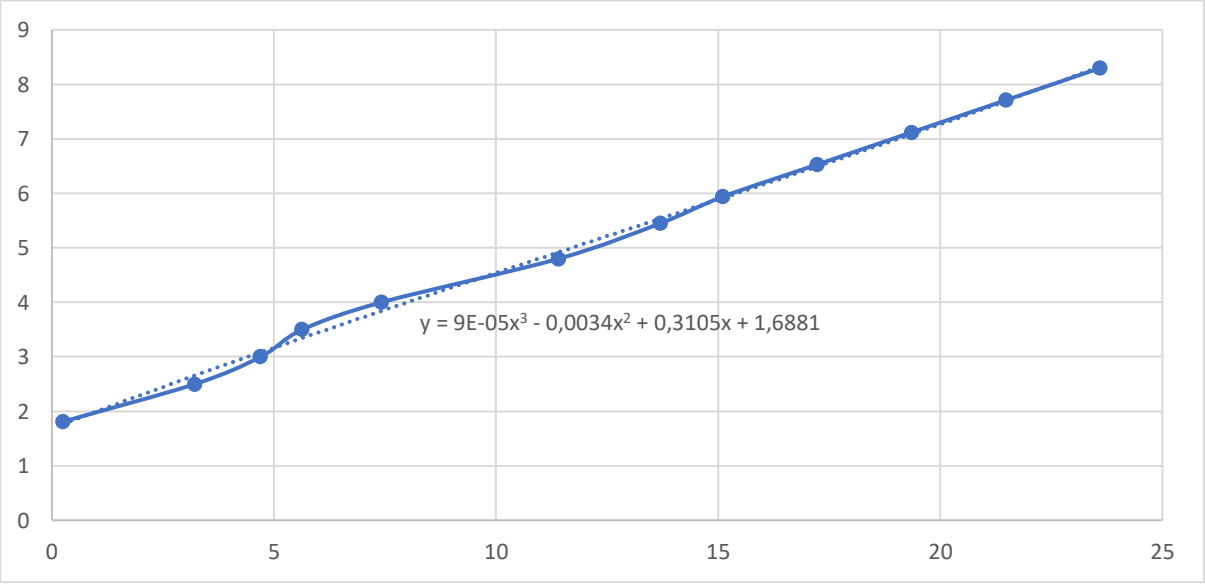
3.2.3 Validación del modelo

Como podemos observar en la Tabla 3.1, los datos medidos son comparados con los datos reales que observamos en el tanque, utilizando los datos medidos como datos para nuestro eje x y los datos reales para nuestro eje y podemos sacar la ecuación que utilizaremos para obtener nuestra función de transferencia vista en la figura 3.10

x	y
Medido	Real
23,6052381	8,3
21,4820952	7,71
19,3589524	7,12
17,2358095	6,53
15,1126667	5,94
13,7	5,45
11,41	4,8
7,42	4
5,64	3,5
4,7	3
3,22	2,5
0,25066667	1,81
Volumen sensor	Volumen Balde real

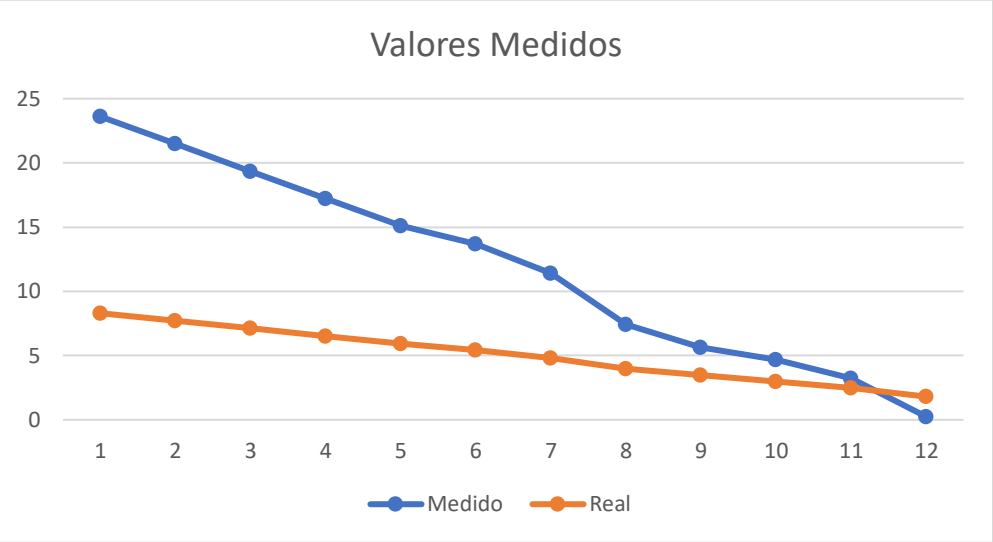
Tabla 3.1 Validación del modelo

Autores: Saul Carvallo & Jonathan Zambrano



Grafica. 3.1 Validación del modelo

Autores: Saul Carvallo & Jonathan Zambrano



Grafica 3.2 Valores medidos

Autores: Saul Carvallo & Jonathan Zambrano

3.3 Graficas obtenidas con controlador PID

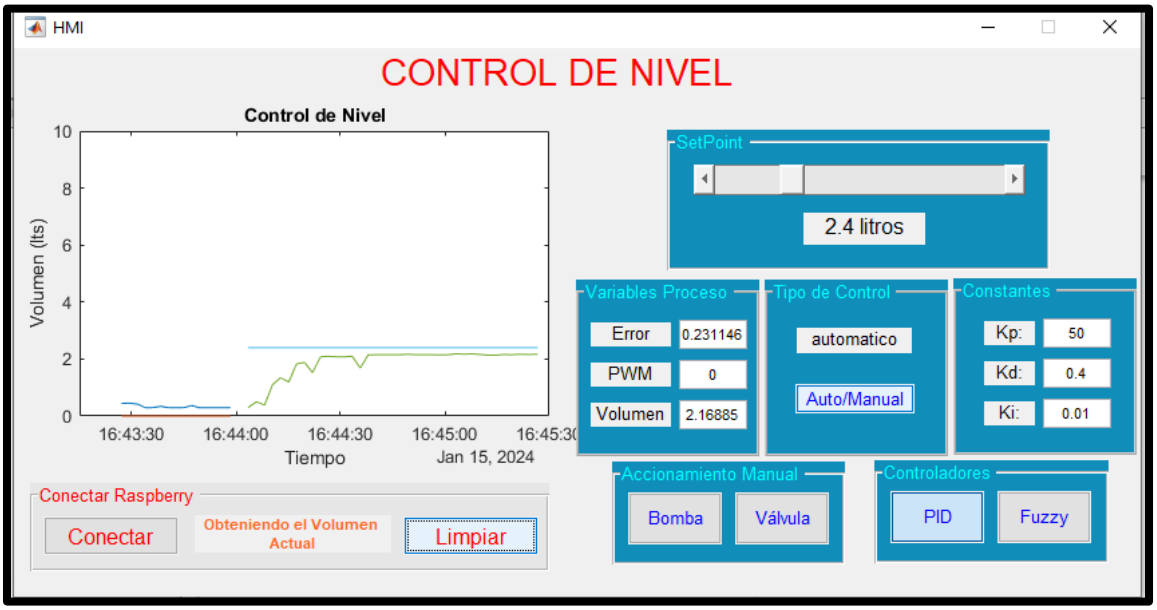


Figura 3.11 Control de nivel usando controlador PID con error de 0.231146

Autores: Saul Carvallo & Jonathan Zambrano

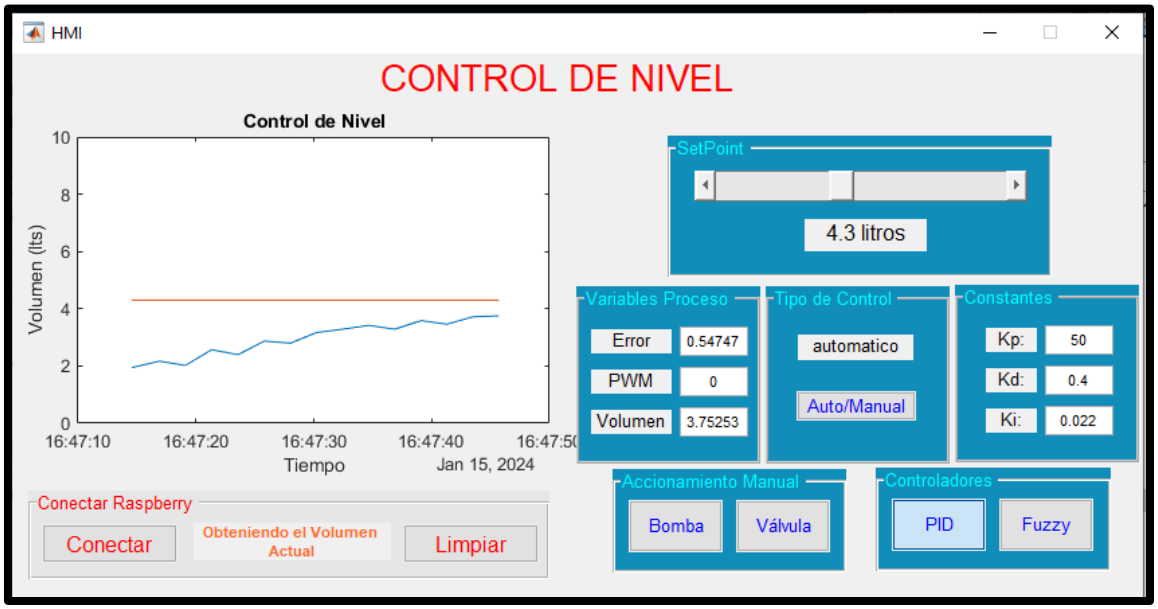


Figura 3.12 Control de nivel usando controlador PID con error de 0.54747

Autores: Saul Carvallo & Jonathan Zambrano

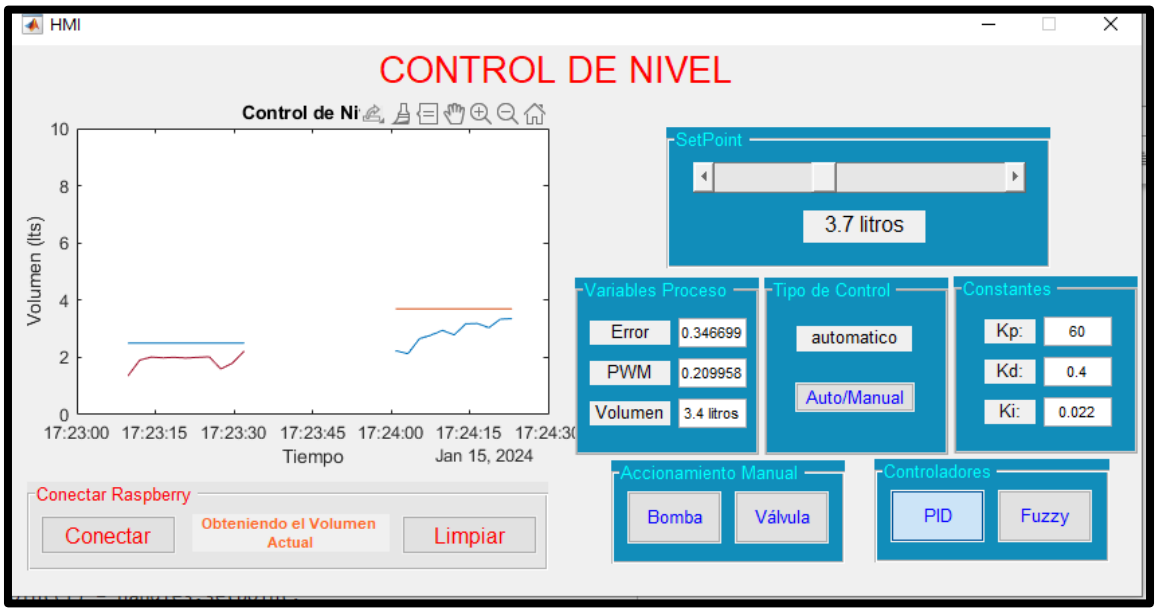


Figura 3.13 Control de nivel usando controlador PID con error de 0.346699

Autores: Saul Carvallo & Jonathan Zambrano

3.4 Graficas obtenidas con controlador Fuzzy

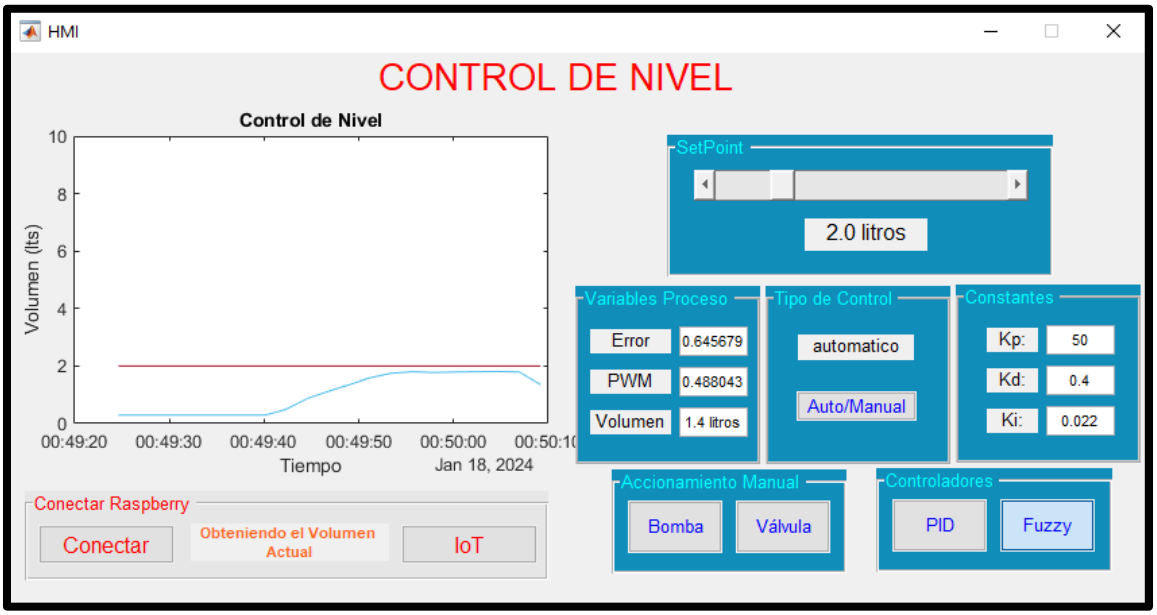


Figura 3.14 Control de nivel usando controlador Fuzzy con error de 0.645679

Autores: Saul Carvallo & Jonathan Zambrano

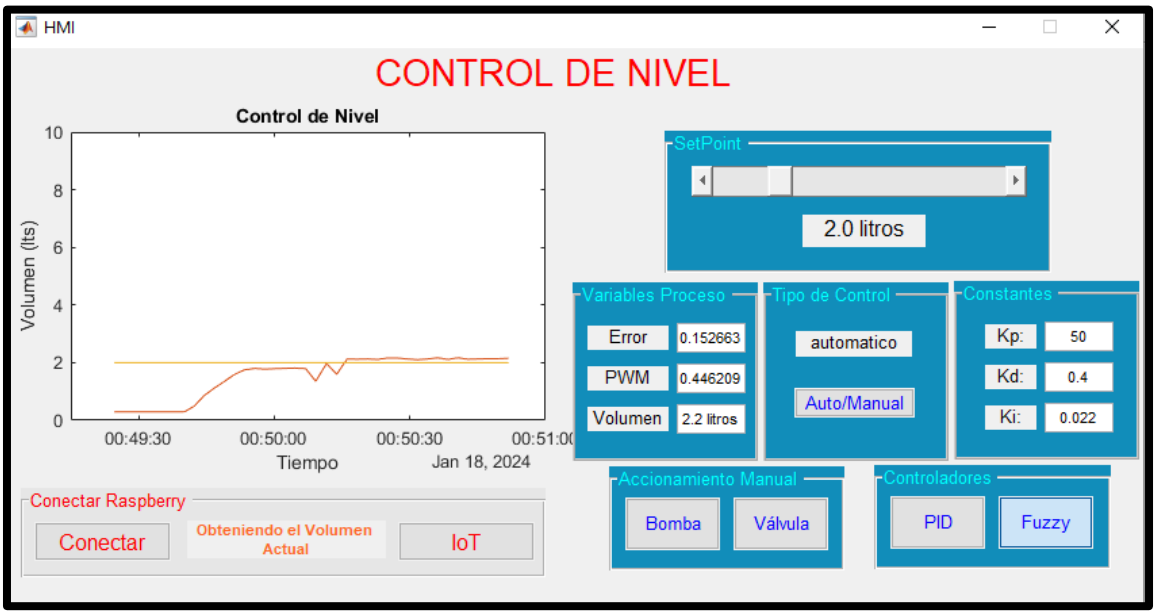


Figura 3.15 Control de nivel usando controlador Fuzzy con error de 0.152663

Autores: Saul Carvallo & Jonathan Zambrano

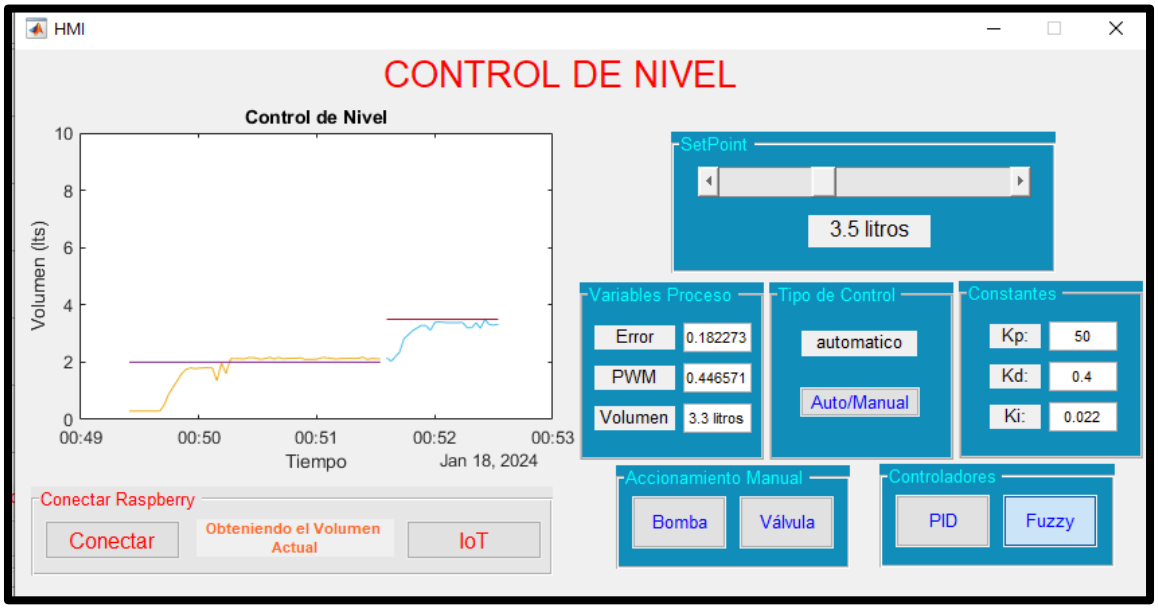


Figura 3.16 Control de nivel usando controlador Fuzzy con error de 0.182273

Autores: Saul Carvallo & Jonathan Zambrano

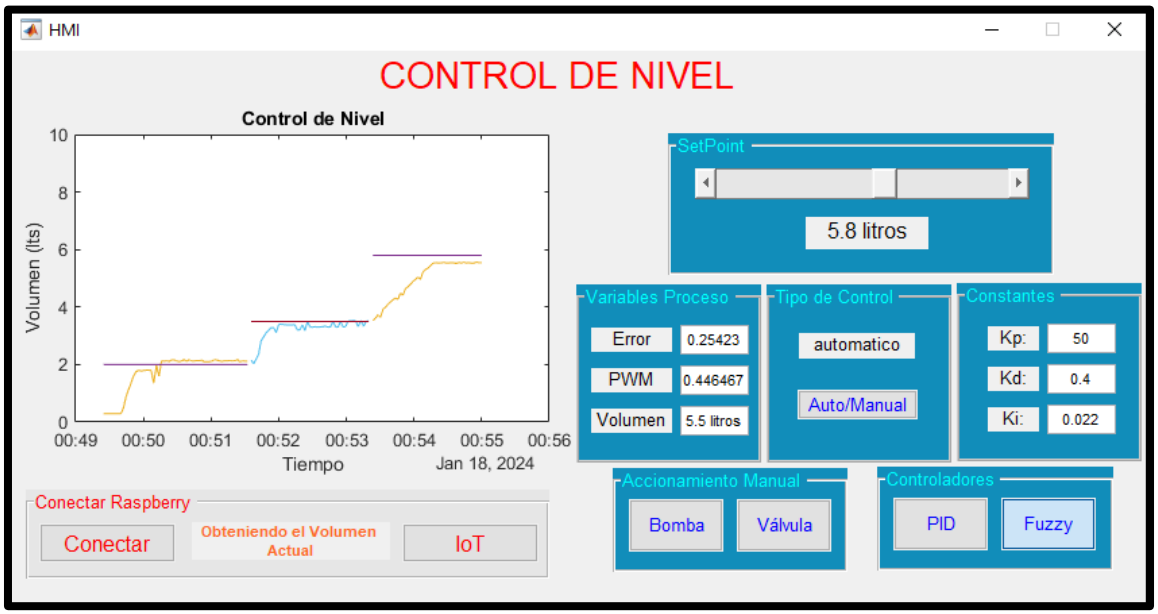


Figura 3.17 Control de nivel usando controlador Fuzzy con error de 0.25423

Autores: Saul Carvallo & Jonathan Zambrano

3.5 Interfaz obtenida en la pantalla

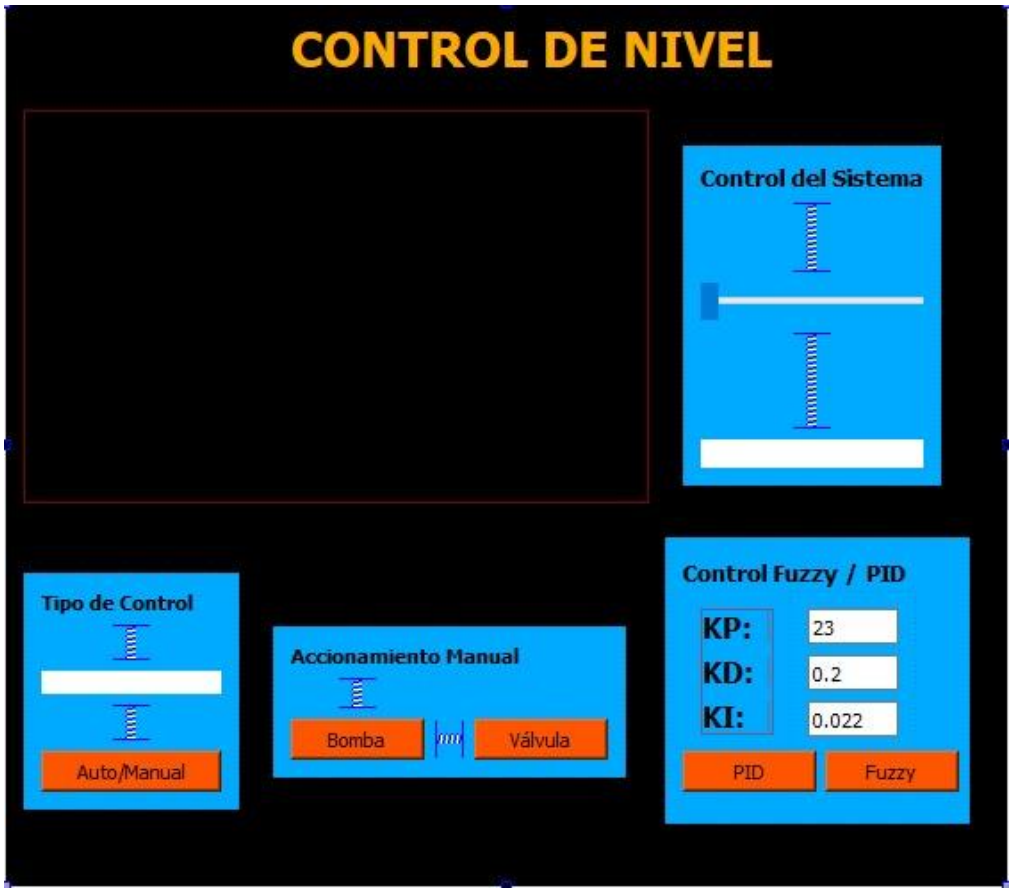


Figura. 3.18 Interfaz de usuario del prototipo embebido utilizando Qt

Autores: Saul Carvallo & Jonathan Zambrano

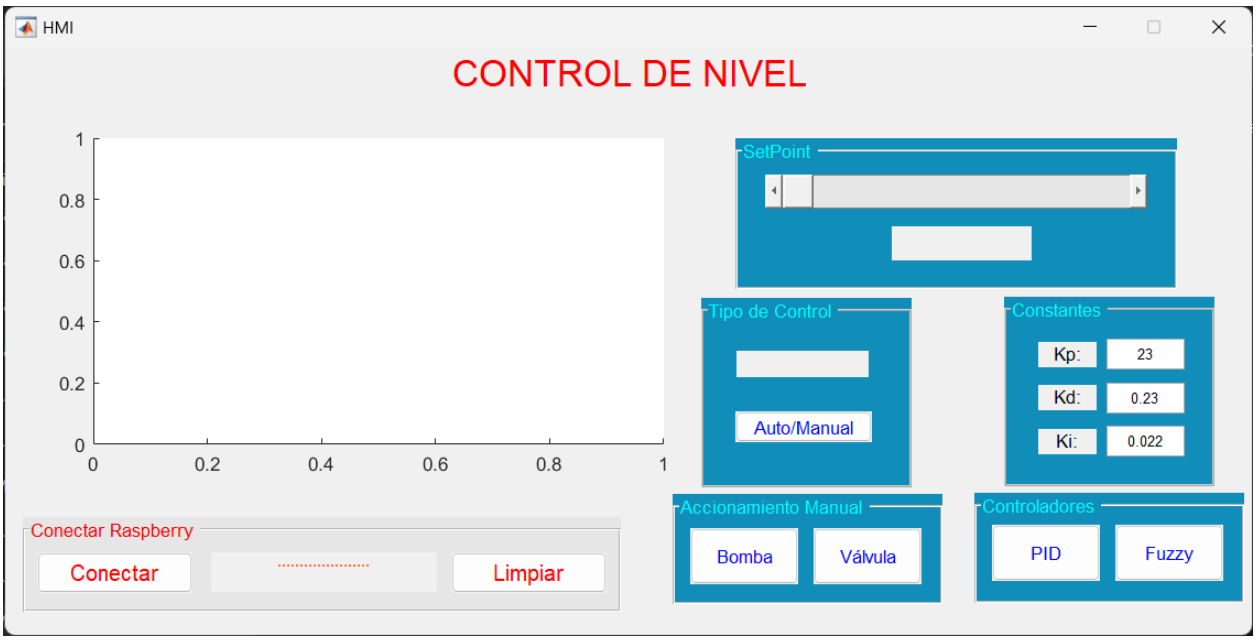


Figura 3.19 Interfaz de usuario del prototipo embebido utilizando Matlab

Autores: Saul Carvallo & Jonathan Zambrano

3.6 Programación del controlador PID en Python

Dentro de la programación que corresponde al controlador PID que podemos encontrar en el punto 1.2 de los anexos, tenemos varias funciones que dan paso al funcionamiento de este.

```
ui = loadUi("/home/nivel/Documentos/Control Nivel.ui", self)
```

Luego de llamar a la clase MyWindow, procedemos a cargar la interfaz gráfica desde el archivo de tipo .ui, de tal manera que se puedan conectar los botones del hmi a las funciones específicas correctamente, como podemos observar en el fragmento de código mostrado.

```
self.auto_manual.clicked.connect(self.cambio_control)
        self.PID.clicked.connect(self.elegir_pid)
        self.Fuzzy.clicked.connect(self.elegir_fuzzy)
        self.bomba.clicked.connect(self.activar_bomba)
        self.valvula.clicked.connect(self.activar_valvula)
```

Tenemos cuatro tipos de modos para inicializar el controlador, los cuales son: manual, automático, pid y fuzzy. La forma de seleccionarlo lo establecemos en las siguientes líneas de código.

```
self.modo = 'manual'
        self.bomba_estado='0'
        self.valvula_estado = '0'
        self.controlador = None
```

Mostramos el setpoint establecido con la función show

```
self.show()
```

Ahora establecemos la función para mandar a activar la bomba, mediante el siguiente fragmento de código.

```
def activar_bomba(self):
        if self.modo == 'manual':
            if self.bomba_estado == '1':
                self.bomba_estado = '0'
                rojo.ChangeDutyCycle(100)

            else:
                self.bomba_estado = '1'
                rojo.ChangeDutyCycle(0)
```

Simplemente, se trata de una condicional donde elegimos el tipo de modo para inicializar el controlador, luego preguntamos si el estado de la bomba es verdadero para luego cambiarlo a falso con un dutycycle de 100, y en caso de que no cambie de estado el dutycycle disminuirá a 0.

Luego de activar la bomba procedemos a activar la vávula que permitirá el flujo de la corriente de agua por la manguera, con el siguiente fragmento de código.

```
def activar_valvula(self):
        if self.modo == 'manual':
            if self.valvula_estado == '1':
                self.valvula_estado = '0'
                GPIO.output(rele, 1)  # Enciende el rele

            else:
                self.valvula_estado = '1'
                GPIO.output(rele, 0)  # Enciende el rele
```

De igual forma se trata de una condicional donde preguntamos el modo de operación de la planta, para posteriormente preguntar el estado de la válvula, dando paso al encendido del relé que vendría a tener la función de interruptor.

Para proceder al cambio de modo en el sistema de manual a automático establecemos la función cambio_control, donde mediante una condicional pasamos de control manual a automático.

```
def cambio_control(self):
        # Cambiar entre 'auto' y 'manual'
        if self.modo == 'automatico':
            self.modo = 'manual'
            self.controlador = None
        else:
            self.modo = 'automatico'

        # Actualizar el texto del label segun el estado actual
        self.tipo_control.setText(f"{self.modo.capitalize()}")
```

El texto que se mostrará en el hmi se actualizará mediante la función tipo_control.setText, donde el valor que se almacene en self.modo se verá mostrado en letras mayúsculas utilizando la función .capitalize().

Ahora procedemos a elegir el PID para nuestra planta, definiendo la función elegir_pid, donde llamamos a las constantes kp, kd, ki que definimos en el inicio del código.

```
def elegir_pid(self):
        if self.modo == 'automatico':
            self.controlador = 'pid'
            print("Seleccionado controlador PID")
            kp = float(self.Kp.text())
            kd = float(self.Kd.text())
            ki = float(self.Ki.text())

            self.setpoint_distance = self.Slider_setpoint.value()
            print(f"Valores Kp: {kp}, Kd: {kd}, Ki: {ki}")
            print(f"Valor del slider PID: {self.setpoint_distance}")
```

También establecimos unos contadores para las variables de error y la integral, como observamos en el fragmento de código mostrado a continuación.

```
# PID variables
            previous_error = 0
            integral = 0

            try:
                while True:

                    GPIO.output(TRIG, GPIO.LOW)
                    time.sleep(0.005)  # Reduzco el tiempo de espera

                    GPIO.output(TRIG, GPIO.HIGH)
                    time.sleep(0.00001)
                    GPIO.output(TRIG, GPIO.LOW)

                    start_time = time.time()
                    end_time = time.time()

                    while GPIO.input(ECHO) == GPIO.LOW and (end_time -
start_time) < 0.1:
                        start_time = time.time()

                    while GPIO.input(ECHO) == GPIO.HIGH and (end_time -
start_time) < 0.1:
                        end_time = time.time()

                    duration = end_time - start_time
                    distancia = (duration * 34300) / 2
                    distancia2 = 23-distancia
                    if distancia2 <=0 :
                        distancia2 = 0

                    volumen2 = ((math.pi * (21.5)**2 * distancia2)/1000)
                    print("Volumen2: %.2f lts" % volumen2)
                    x=volumen2
```

```
                    volumen = 0.0023 * x**3 - 0.0668 * x**2 + 0.8638 * x + 0.2938
                    if volumen <=0 :
                        volumen = 0

                    print("Volumen: %.2f lts" % volumen)
                    print("SetPoint: %.2f lts" % self.setpoint_distance)

                    # Control PID
                    error = self.setpoint_distance - volumen
                    integral += error
                    derivative = error - previous_error

                    pid_output = kp * error + ki * integral + kd * derivative
                    pid_output = max(0, min(100, pid_output))  # Satura la salida
entre 0 y 100

                    rojo.ChangeDutyCycle(pid_output)
                    print("PWM: %.2d " % pid_output)

                    # Control ON/OFF basado en el setpoint
                    if volumen < self.setpoint_distance:
                        GPIO.output(rele, 1)  # Enciende el rele
                    else:
                        GPIO.output(rele, 0)  # Apaga el rele

                    previous_error = error
                    distancia_data.append(volumen)
                    pid_output_data.append(pid_output)
                    setpoint_data.append(self.setpoint_distance)

                    time.sleep(0.05)  # Reduzco el tiempo de
espera

            finally:
                GPIO.cleanup()


        else:
            print("No estas en modo automatico")
```

Esta parte del código se encarga de implementar el control PID para mantener el nivel de líquido en un tanque en un valor deseado (setpoint).

Verificamos que el sistema esté en modo automático (self.modo == 'automatico') antes de proceder. Posterior a eso se establece el controlador actual como 'pid' (self.controlador = 'pid').

Obtenemos los valores de los parámetros del PID (constantes proporcionales, integrales y derivativas) y el setpoint desde la interfaz gráfica.
Se inicializan las variables PID, como previous_error e integral, que son necesarias para el control PID.
Se inicia un bucle while True que realiza la medición de distancia y control PID de forma continua.
Dentro del bucle, se realiza la medición de distancia con un sensor ultrasónico y se calcula el volumen del líquido en el tanque.
Se calcula el error en relación al setpoint, y se aplica el control PID para obtener la salida (pid_output).
Se limita la salida PID entre 0 y 100.
La salida PID se utiliza para controlar el ciclo de trabajo de un LED simulando un actuador.
Se ajusta un relé en función del nivel de líquido en relación con el setpoint.
Se actualizan las variables y las listas de datos para su posterior análisis.
Se incluye una pausa de 0.05 segundos entre cada iteración para reducir el tiempo de espera.

3.7 Programación del controlador PID y Fuzzy en Matlab

La programación final se realizó en Matlab, por fallos en la compatibilidad de librerías usando Python, a continuación, una breve explicación del código que encontramos en el punto 1.3 de los anexos:

Funciones Principales:

- HMI_OpeningFcn: Se ejecuta al abrir la interfaz gráfica. Inicializa las variables y configura los pines GPIO.
- HMI_OutputFcn: Devuelve la salida de la función principal a la línea de comandos.
- conectar_Callback: Intenta establecer una conexión con una Raspberry Pi y configura los pines GPIO.

Interfaz Gráfica:

- La GUI está creada con GUIDE, proporcionando botones, sliders y otros controles para interactuar con el sistema.

Callbacks de Botones y Sliders:

- setpoint_Callback: Se ejecuta al mover el slider de setpoint. Actualiza la variable setpoint y muestra el valor en la interfaz.
- bomba_Callback y valvula_Callback: Controlan las bombas y válvulas en el modo manual. Realizan mediciones y grafican los datos.
- auto_manual_Callback: Cambia entre modos automático y manual.
- boton_pid_Callback: Selecciona el controlador PID y realiza el control de nivel utilizando PID.
- conectar_Callback: Intenta establecer una conexión con una Raspberry Pi y configura los pines GPIO.

Función de Medición de Distancia (measureDistance):

- Utiliza el sensor ultrasónico conectado a los pines trigPin y echoPin para medir la distancia al nivel del líquido.
- Calcula el volumen basado en la distancia medida.
- Aplica un filtro de media móvil para suavizar las lecturas.

Control PID (boton_pid_Callback):

- Selecciona el controlador PID y realiza un bucle infinito para controlar el nivel del líquido.
- Utiliza los valores de ganancia kp, ki y kd introducidos por el usuario.
- Controla la bomba y la válvula según el error entre el setpoint y el volumen actual.

Gráficos (axes1):

- Muestra un gráfico en tiempo real de la evolución del nivel del líquido.

Variables de Estado:

- modo: Puede ser 'automatico' o 'manual'.
- bomba_estado y valvula_estado: Representan el estado activado/desactivado de la bomba y la válvula.
- controlador: Puede ser 'None', 'pid' o 'fuzzy'.
- setpoint: Almacena el valor deseado del nivel del líquido.

3.8 Comparación entre programación con Python vs Matlab

En la tabla 3.2 podemos observar la comparativa con calificaciones que van del 1 al 10 (siendo 1 el menos favorable y el 10 el más favorable) entre la programación de nuestra planta de fluidos usando Python vs usando Matlab, concluyendo así que la mejor opción a usar es la de Matlab.

Interprete	Compatibilidad de librerías	Interfaz	Tratamiento de datos	Comunicación Wifi	Precisión de los datos
Python	5	8	8	7	9
Matlab	10	10	9	9	9

Tabla 3.2 Comparativa entre programación Python vs Matlab

Autores: Saul Carvallo & Jonathan Zambrano

3.9 Conexión a la nube con IoT

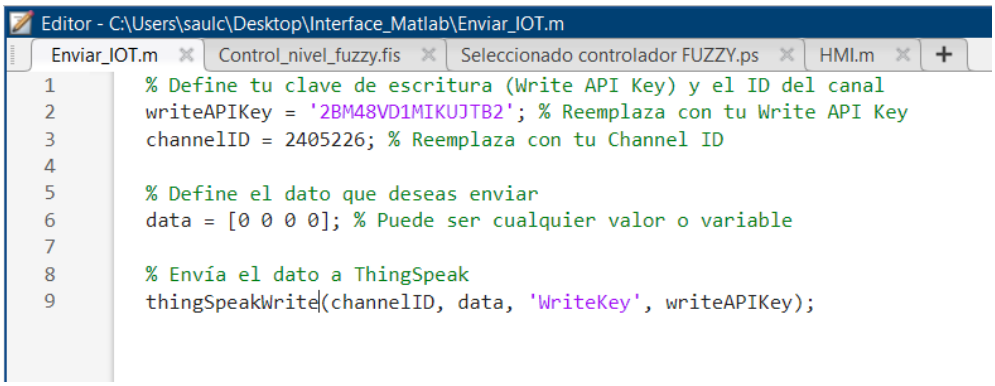


Figura 3.20 Conexión a la nube con IoT

Autores: Saul Carvallo & Jonathan Zambrano

Con este código vamos a enviar los datos a ThingSpeak, una plataforma en la nube para la recolección y visualización de datos en tiempo real. A continuación, se explica paso a paso:

1. 'writeAPIKey' y 'channelID': Estos son dos parámetros cruciales para autenticar y dirigir el envío de datos al canal correcto en ThingSpeak.

- 'writeAPIKey': Es una clave que autentica la escritura en el canal de ThingSpeak. Deberías tener esta clave asociada con tu cuenta de ThingSpeak y el canal específico al que deseas enviar datos.

- 'channelID': Es el identificador único del canal de ThingSpeak al que se enviarán los datos.

2. ‘data’: Es un vector que contiene los datos que deseas enviar al canal de ThingSpeak. En este caso, ‘data’ está inicializado con ‘[0 0 0 0]’. Puedes cambiar estos valores según los datos específicos que quieras enviar.

3. ‘thingSpeakWrite’: Es una función proporcionada por la Toolbox de ThingSpeak en MATLAB que facilita el envío de datos a un canal de ThingSpeak.

- ‘channelID’: El ID del canal al que se enviarán los datos.
- ‘data’: Los datos que se enviarán al canal.
- ‘‘WriteKey‘‘: Parámetro adicional que especifica la clave de escritura asociada al canal.

En resumen, este código envía el vector ‘[0 0 0 0]’ al canal de ThingSpeak con el ID ‘channelID’ y la clave de escritura ‘writeAPIKey’.

3.10 Visualización de datos en la nube

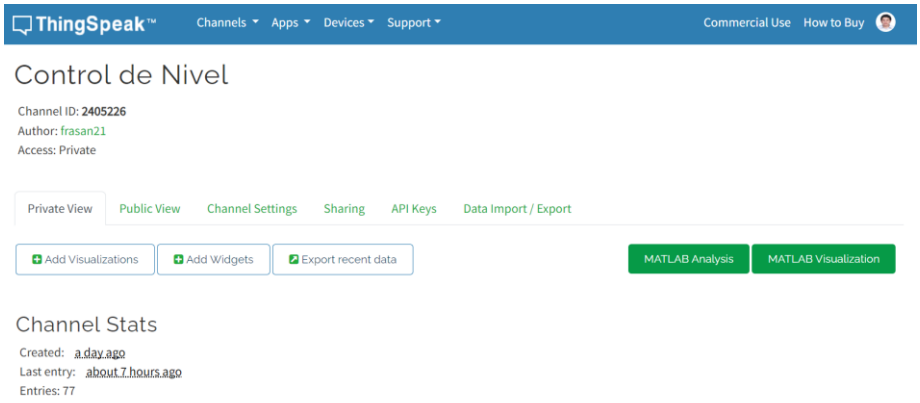


Figura 3.21 Dashboard creado en ThingSpeak

Autores: Saul Carvallo & Jonathan Zambrano

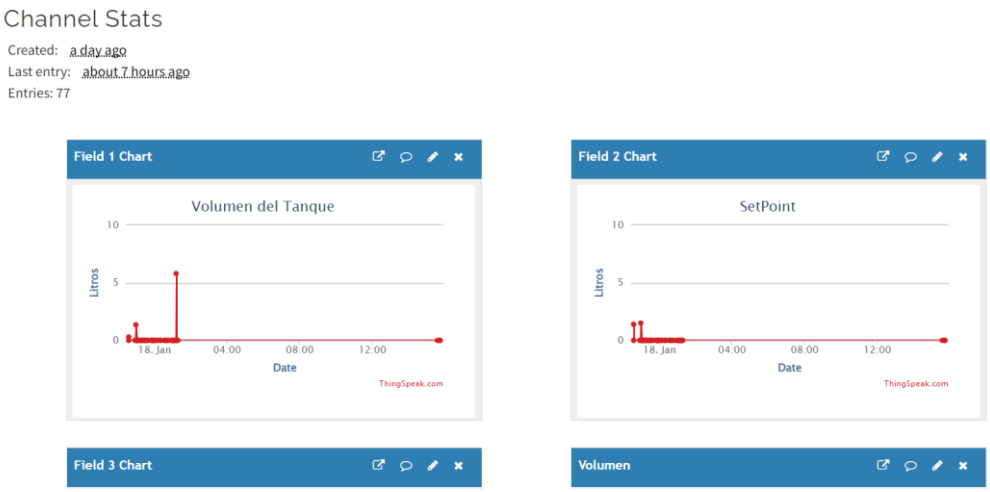


Figura 3.22 Datos almacenados del volumen del tanque y setpoint

Autores: Saul Carvallo & Jonathan Zambrano

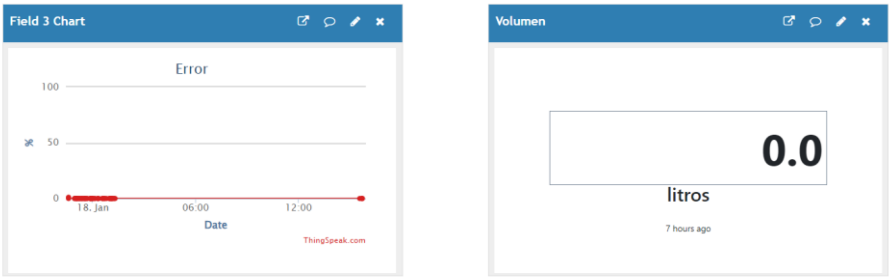


Figura 3.23 Datos del error y volumen

Autores: Saul Carvallo & Jonathan Zambrano

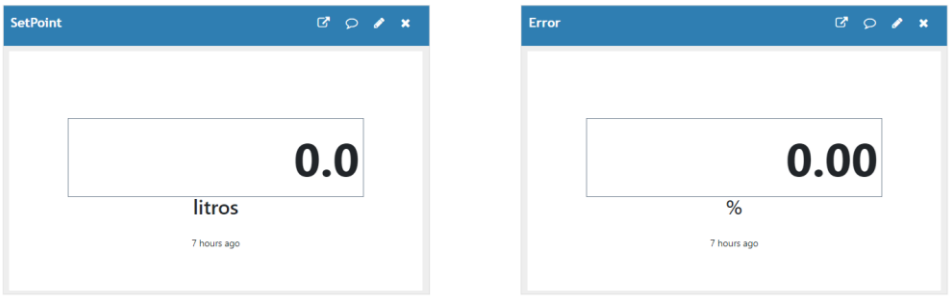


Figura 3.24 Datos del error y litros de agua en el tanque

Autores: Saul Carvallo & Jonathan Zambrano

CAPÍTULO 4

4 CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- En base a lo desarrollado se concluye que se creó y se validó el diseño mecánico con sus respectivas normas (coeficiente de seguridad), con esto se logró dar soporte a la carga en general de los tanques, además de diseñar de manera oportuna el sistema eléctrico/electrónico para la planta de fluidos con sus respectivas protecciones, para poder ejecutar el control de nivel.
- Se implementó de manera satisfactoria el prototipo a escala de la planta de fluidos para el Laboratorio de sistemas de control, teniendo como medidas finales de diseño de 61.3 cm de ancho,82.68 cm de alto y 55.7 cm de profundidad.
- En base al desarrollo integral del proyecto se optó por la realización de dos versiones de programación en distintos entornos, como lo son Python y Matlab, dando como mejor resultado la segunda opción, esto debido a que la compatibilidad de las librerías en Python resultó fallida, entonces se consideró como la opción más viable para el funcionamiento de los controladores PID y Fuzzy, dicha comparación la podemos observar en la tabla 2.9.
- En conclusión, Luego de realizar las pruebas respectivas tanto para el controlador PID como para el Fuzzy, concluimos que el controlador óptimo fue el Fuzzy, ya que su tiempo de estabilización fue menor, así como su margen de error, teniendo unos índices de desempeño más favorables que los del controlador PID.
- En la configuración de los controladores PID y Fuzzy como podemos observar desde la Figura 3.11 hasta la Figura 3.17, se seleccionaron valores específicos para las variables Kp, Kd y Ki, ajustados para un intervalo de volumen de 2 a 5.8 litros. Aunque ambos controladores demostraron ser eficaces, el controlador Fuzzy mostró un menor error promedio (0.326871) y un PWM promedio de 0.460274. Estos resultados sugieren un rendimiento superior en comparación con el controlador PID, que presentó un error promedio de 0.375092.
- No obstante, los datos fueron subidos a la nube en el entorno de ThingSpeak, para observar los mismos en tiempo real junto a las gráficas, dando paso a un posible uso de ellos para otros proyectos a futuro.

4.2 Recomendaciones

- Tomar en cuenta los coeficientes de seguridad si se desea agregar nuevos dispositivos o cambiar de estructura mecánica, para evitar fallos en la estabilidad y precisión de la obtención de los datos de nuestra planta.
- A pesar de que se decidió por usar Matlab para la programación, se recomienda seguir monitoreando las actualizaciones y mejoras en el entorno de Python para evaluar la posibilidad de emigrar de plataforma de programación.
- Continuar monitoreando y ajustando el controlador PID para obtener un mejor desempeño, aunque se haya concluido que el controlador Fuzzy es el más ideal.
- Se recomienda asegurar que los datos almacenados en la nube usando la plataforma ThingSpeak estén seguros y protegidos para evitar posibles pérdidas de información.
- Es recomendable seguir identificando posibles mejoras en áreas específicas de nuestra planta para obtener un mejor provecho de ella en otros ámbitos.

ANEXOS

1.1. CÓDIGO PARA EL CONTROL DE NIVEL

```
# Importamos la paqueteria necesaria
import RPi.GPIO as GPIO
import time

TRIG = 17 #Variable que contiene el GPIO al cual conectamos la serial TRIG del sensor
ECHO = 27 #Variable que contiene el GPIO al cual conectamos la serial ECHO del sensor
rele = 22

GPIO.setmode(GPIO.BCM)     #Establecemos el modo segï¿½n el cual nos referiremos a los GPIO de nuestra RPi
GPIO.setup(TRIG, GPIO.OUT) #Configuramos el pin TRIG como una salida
GPIO.setup(ECHO, GPIO.IN)  #Configuramos el pin ECHO como una salida

#Variables para el PWM
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

GPIO.setup(24, GPIO.OUT)
rojo = GPIO.PWM(24, 100)
rojo.start(100)

GPIO.setup(rele, GPIO.OUT)

#Contenemos el cï¿½digo principal en un aestructura try para limpiar los GPIO al terminar o presentarse un error
try:
   #Implementamos un loop infinito
   while True:

      # Ponemos en bajo el pin TRIG y despuï¿½s esperamos 0.5 seg para que el transductor se estabilice
      GPIO.output(TRIG, GPIO.LOW)
      time.sleep(0.5)

      #Ponemos en alto el pin TRIG esperamos 10 uS antes de ponerlo en bajo
      GPIO.output(TRIG, GPIO.HIGH)
      time.sleep(0.00001)
      GPIO.output(TRIG, GPIO.LOW)

      # En este momento el sensor envï¿½a 8 pulsos ultrasï¿½nicos de 40kHz y coloca su pin ECHO en alto
      # Debemos detectar dicho evento para iniciar la mediciï¿½n del tiempo

      while True:
         pulso_inicio = time.time()
         if GPIO.input(ECHO) == GPIO.HIGH:
            break

      # El pin ECHO se mantendrï¿½ en HIGH hasta recibir el eco rebotado por el obstï¿½culo.
      # En ese momento el sensor pondrï¿½ el pin ECHO en bajo.
           # Prodedemos a detectar dicho evento para terminar la mediciï¿½n del tiempo

      while True:
         pulso_fin = time.time()
         if GPIO.input(ECHO) == GPIO.LOW:
            break

      # Tiempo medido en segundos
      duracion = pulso_fin - pulso_inicio

      #Obtenemos la distancia considerando que la seï¿½al recorre dos veces la distancia a medir y que la velocidad
del sonido es 343m/s
```

```
        distancia = (34300 * duracion) / 2

        GPIO.output(rele,0)
        # Imprimimos resultado
        print( "Distancia: %.2f cm" % distancia)
        for i in range(100,-1,-1):
            rojo.ChangeDutyCycle(100 - i)
            time.sleep(0.05)

        print("Ciclo completo")
        GPIO.output(rele,1)
        time.sleep(2)

finally:
    # Reiniciamos todos los canales de GPIO.
    GPIO.cleanup()
```

1.2. PROGRAMACIÓN DEL CONTROLADOR PID EN PYTHON

```python
import RPi.GPIO as GPIO
import time
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow
from PyQt5.uic import loadUi
import math
import matplotlib.pyplot as plt
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas
from PyQt5.QtCore import Qt  # Importar Qt directamente
import numpy as np

#Variables de los actuadores y sensores
TRIG = 17
ECHO = 27
rele = 22
led_pin = 24

# Listas para almacenar los datos de las senales
distancia_data = []
pid_output_data = []
setpoint_data = []

GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)
GPIO.setup(rele, GPIO.OUT)
GPIO.setup(led_pin, GPIO.OUT)

rojo = GPIO.PWM(led_pin, 100)
rojo.start(0)

# PID parameters (tune these according to your system)
kp = 23
ki = 0.2
kd = 0.02

class PlotWidget(FigureCanvas):
    def __init__(self, parent=None):
        self.fig, self.ax = plt.subplots(facecolor='gray')
        super().__init__(self.fig)
        self.setParent(parent)
        self.ax.grid()
```

```
        self.ax.margins(x=0)

    def update_plot(self, x_data, y_data, xlabel, ylabel, title):
        self.ax.clear()
        self.ax.plot(x_data, y_data, marker='o')
        self.ax.set_xlabel(xlabel)
        self.ax.set_ylabel(ylabel)
        self.ax.set_title(title)
        self.draw()


class MyWindow(QMainWindow):

    def __init__(self):
        super(MyWindow, self).__init__()
        # Cargar la interfaz grafica desde el archivo .ui
        ui = loadUi("/home/nivel/Documentos/Control Nivel.ui", self)

        # Conectar los botones a las funciones correspondientes
        self.auto_manual.clicked.connect(self.cambio_control)
        self.PID.clicked.connect(self.elegir_pid)
        self.Fuzzy.clicked.connect(self.elegir_fuzzy)
        self.bomba.clicked.connect(self.activar_bomba)
        self.valvula.clicked.connect(self.activar_valvula)


        # Inicializar el estado del modo y el controlador (puede ser 'manual', 'auto', 'pid' o
'fuzzy')
        self.modo = 'manual'
        self.bomba_estado='0'
        self.valvula_estado = '0'
        self.controlador = None
        #setpoint
        self.show()
    def activar_bomba(self):
        if self.modo == 'manual':
            if self.bomba_estado == '1':
                self.bomba_estado = '0'
                rojo.ChangeDutyCycle(100)

            else:
                self.bomba_estado = '1'
                rojo.ChangeDutyCycle(0)

    def activar_valvula(self):
        if self.modo == 'manual':
            if self.valvula_estado == '1':
                self.valvula_estado = '0'
                GPIO.output(rele, 1)  # Enciende el rele

            else:
                self.valvula_estado = '1'
                GPIO.output(rele, 0)  # Enciende el rele

    def cambio_control(self):
        # Cambiar entre 'auto' y 'manual'
        if self.modo == 'automatico':
            self.modo = 'manual'
            self.controlador = None
        else:
            self.modo = 'automatico'

        # Actualizar el texto del label segun el estado actual
        self.tipo_control.setText(f"{self.modo.capitalize()}")
```

```
    def elegir_pid(self):
        if self.modo == 'automatico':
            self.controlador = 'pid'
            print("Seleccionado controlador PID")
            kp = float(self.Kp.text())
            kd = float(self.Kd.text())
            ki = float(self.Ki.text())

            self.setpoint_distance = self.Slider_setpoint.value()
            print(f"Valores Kp: {kp}, Kd: {kd}, Ki: {ki}")
            print(f"Valor del slider PID: {self.setpoint_distance}")

            # PID variables
            previous_error = 0
            integral = 0

            try:
                while True:

                    GPIO.output(TRIG, GPIO.LOW)
                    time.sleep(0.005)  # Reduzco el tiempo de espera

                    GPIO.output(TRIG, GPIO.HIGH)
                    time.sleep(0.00001)
                    GPIO.output(TRIG, GPIO.LOW)

                    start_time = time.time()
                    end_time = time.time()

                    while GPIO.input(ECHO) == GPIO.LOW and (end_time - start_time) < 0.1:
                        start_time = time.time()

                    while GPIO.input(ECHO) == GPIO.HIGH and (end_time - start_time) < 0.1:
                        end_time = time.time()

                    duration = end_time - start_time
                    distancia = (duration * 34300) / 2
                    distancia2 = 23-distancia
                    if distancia2 <=0 :
                        distancia2 = 0

                    volumen2 = ((math.pi * (21.5)**2 * distancia2)/1000)
                    print("Volumen2: %.2f lts" % volumen2)
                    x=volumen2
                    volumen = 0.0023 * x**3 - 0.0668 * x**2 + 0.8638 * x + 0.2938
                    if volumen <=0 :
                        volumen = 0

                    print("Volumen: %.2f lts" % volumen)
                    print("SetPoint: %.2f lts" % self.setpoint_distance)

                    # Control PID
                    error = self.setpoint_distance - volumen
                    integral += error
                    derivative = error - previous_error

                    pid_output = kp * error + ki * integral + kd * derivative
                    pid_output = max(0, min(100, pid_output))  # Satura la salida entre 0 y
100


                    rojo.ChangeDutyCycle(pid_output)
                    print("PWM: %.2d " % pid_output)

                    # Control ON/OFF basado en el setpoint
                    if volumen < self.setpoint_distance:
```

```
                        GPIO.output(rele, 1)  # Enciende el rele
                    else:
                        GPIO.output(rele, 0)  # Apaga el rele

                    previous_error = error
                    distancia_data.append(volumen)
                    pid_output_data.append(pid_output)
                    setpoint_data.append(self.setpoint_distance)

                    time.sleep(0.05)  # Reduzco el tiempo de espera

            finally:
                GPIO.cleanup()


        else:
            print("No estas en modo automatico")

    def elegir_fuzzy(self):
        if self.modo == 'automatico':
            self.controlador = 'fuzzy'
            print("Seleccionado controlador FUZZY")
        else:
            print("No estas en modo automatico")

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MyWindow()
    sys.exit(app.exec_())
```

1.3. PROGRAMACIÓN DEL CONTROLADOR PID Y FUZZY EN MATLAB

```
function varargout = HMI(varargin)
% HMI MATLAB code for HMI.fig
%      HMI, by itself, creates a new HMI or raises the existing
%      singleton*.
%
%      H = HMI returns the handle to a new HMI or the handle to
%      the existing singleton*.
%
%      HMI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in HMI.M with the given input arguments.
%
%      HMI('Property','Value',...) creates a new HMI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before HMI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to HMI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help HMI

% Last Modified by GUIDE v2.5 13-Jan-2024 04:34:05

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @HMI_OpeningFcn, ...
                   'gui_OutputFcn',  @HMI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
```

```
                'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% --- Executes just before HMI is made visible.
function HMI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to HMI (see VARARGIN)

% Choose default command line output for HMI
handles.output = hObject;

handles.modo = 'automatico'
handles.bomba_estado = 0
handles.valvula_estado = 0
handles.controlador = 'None'

% Configurar los pines GPIO
% Los números de pines en MATLAB son strings
handles.rele = 22;
handles.bomba_pwm = 24;
handles.trigPin = 17;
handles.echoPin = 27;

%Variable del slider
handles.setpoint=0

%Variables de la interfaz
handles.infinito = 1

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes HMI wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = HMI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on slider movement.
function setpoint_Callback(hObject, eventdata, handles)
handles.setpoint = round(get(hObject, 'Value'), 1);
% Actualizar la interfaz de usuario
formattedText = sprintf('%.1f litros', handles.setpoint);
set(handles.valor, 'String', formattedText);


% Guardar los cambios en la estructura handles
guidata(hObject, handles);

% hObject    handle to setpoint (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'Value') returns position of slider
%        get(hObject,'Min') and get(hObject,'Max') to determine range of slider
```

```
% --- Executes during object creation, after setting all properties.
function setpoint_CreateFcn(hObject, eventdata, handles)
% hObject    handle to setpoint (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end


% --- Executes on button press in bomba.
function bomba_Callback(hObject, eventdata, handles)

if strcmp(handles.modo, 'manual')
    if handles.bomba_estado == 1
        % Inicializar vectores para almacenar datos
        numMuestras = 2;  % Número de muestras que deseas recopilar
        datosVolumen = zeros(1, numMuestras);
        datosTiempo = datetime(zeros(1, numMuestras), 0, 0); % Inicializa un array de tipo datetime

        for i = 1:numMuestras
            datosVolumen(i) = measureDistance(handles.mypi, handles.trigPin, handles.echoPin);
            datosTiempo(i) = datetime('now');
            pause(1); % Pausa de 1 segundo entre mediciones
        end

        % Obtener el handle del axes
        axesHandle1 = handles.axes1;  % Asegúrate de ajustar 'axes1' al Tag de tu axes
        % Graficar en el axes
        plot(axesHandle1,datosTiempo,datosVolumen);
        % Configurar propiedades del axes
        title(axesHandle1, 'Control de Nivel');  % Agrega un título
        xlabel(axesHandle1, 'Tiempo');              % Etiqueta del eje X
        ylabel(axesHandle1, 'Volumen (lts)');          % Etiqueta del eje Y
        ylim([0 10])
        handles.bomba_estado = 0
        writePWMDutyCycle(handles.mypi, handles.bomba_pwm, 1);

    else
        % Para detener la señal PWM, puedes usar:
        writePWMDutyCycle(handles.mypi, handles.bomba_pwm, 0);
        handles.bomba_estado = 1

    end
end
guidata(hObject, handles);


% hObject    handle to bomba (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in valvula.
function valvula_Callback(hObject, eventdata, handles)


if strcmp(handles.modo, 'manual')
    if handles.valvula_estado == 1
        % Inicializar vectores para almacenar datos
        numMuestras = 2;  % Número de muestras que deseas recopilar
        datosVolumen = zeros(1, numMuestras);
        datosTiempo = datetime(zeros(1, numMuestras), 0, 0); % Inicializa un array de tipo datetime

        for i = 1:numMuestras
            datosVolumen(i) = measureDistance(handles.mypi, handles.trigPin, handles.echoPin);
            datosTiempo(i) = datetime('now');
            pause(1); % Pausa de 1 segundo entre mediciones
        end

        % Obtener el handle del axes
```

```
        axesHandle1 = handles.axes1;  % Asegúrate de ajustar 'axes1' al Tag de tu axes
        % Graficar en el axes
        plot(axesHandle1,datosTiempo,datosVolumen);
        % Configurar propiedades del axes
        title(axesHandle1, 'Control de Nivel');  % Agrega un título
        xlabel(axesHandle1, 'Tiempo');              % Etiqueta del eje X
        ylabel(axesHandle1, 'Volumen (lts)');          % Etiqueta del eje Y
        ylim([0 10])
        handles.valvula_estado = 0
        writeDigitalPin(handles.mypi, handles.rele, 1);

    else
        %handles.volumen_inicial()
        handles.valvula_estado = 1
        writeDigitalPin(handles.mypi, handles.rele, 0);
    end
end
guidata(hObject, handles);


% hObject    handle to valvula (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in auto_manual.
function auto_manual_Callback(hObject, eventdata, handles)
% Cambiar entre 'auto' y 'manual'
if strcmp(handles.modo, 'automatico')
    handles.modo = 'manual';
    handles.controlador = 'None';
else
    handles.modo = 'automatico';
    % Puedes agregar aquí cualquier configuración adicional para el modo automático
end

% Actualizar la interfaz de usuario
set(handles.control, 'String', handles.modo);

% Guardar los cambios en la estructura handles
guidata(hObject, handles);


% hObject    handle to auto_manual (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in boton_pid.
function boton_pid_Callback(hObject, eventdata, handles)

Datosvolumen = zeros();
Datostiempo = datetime(zeros(), 0, 0); % Inicializa un array de tipo datetime;
DatosSetpoint = zeros();

i=1;
if strcmp(handles.modo, 'automatico')
    handles.controlador = 'pid'
    print("Seleccionado controlador PID")
   % Obtener valores como cadenas de texto
    kp_str = get(handles.kp, 'String');
    kd_str = get(handles.kd, 'String');
    ki_str = get(handles.ki, 'String');

    % Convertir cadenas a valores numéricos
    kp = str2double(kp_str);
    kd = str2double(kd_str);
    ki = str2double(ki_str);

    setpoint_distance = handles.setpoint;
    disp(['Kp: ', num2str(kp)]);
    disp(['Kd: ', num2str(kd)]);
    disp(['Ki: ', num2str(ki)]);
    disp(['Setpoint: ', num2str(setpoint_distance)]);
```

```
    %PID variables
    previous_error = 0
    integral = 0


    while handles.infinito == 1

        volumen = measureDistance(handles.mypi, handles.trigPin, handles.echoPin);
        %Control PID
        error = handles.setpoint - volumen
        integral = integral + error
        derivative = error - previous_error

        pid_output = kp * error + ki * integral + kd * derivative
        pid_output = max(5, min(100, pid_output))  %Satura la salida entre 0 y 100

        writePWMDutyCycle(handles.mypi, handles.bomba_pwm, pid_output/100);


        %Control ON/OFF basado en el setpoint
        if volumen < handles.setpoint
            writeDigitalPin(handles.mypi, handles.rele, 1);% Apaga el rele
        else
            writeDigitalPin(handles.mypi, handles.rele, 0);  % Enciende el rele
        end

        previous_error = error

        if abs(error)<0.15
            writePWMDutyCycle(handles.mypi, handles.bomba_pwm, 0);
            writeDigitalPin(handles.mypi, handles.rele, 1); % Apaga el rele
            handles.infinito = 0
        end

        Datosvolumen(i) = measureDistance(handles.mypi, handles.trigPin, handles.echoPin);
        Datostiempo(i) = datetime('now');
        DatosSetpoint(i) = handles.setpoint;
        % Obtener el handle del axes
        axesHandle1 = handles.axes1;  % Asegúrate de ajustar 'axes1' al Tag de tu axes
        % Graficar en el axes
        plot(axesHandle1,Datostiempo,Datosvolumen);
        hold on
        plot(axesHandle1,Datostiempo,DatosSetpoint);
        % Configurar propiedades del axes
        title(axesHandle1, 'Control de Nivel');  % Agrega un título
        xlabel(axesHandle1, 'Tiempo');              % Etiqueta del eje X
        ylabel(axesHandle1, 'Volumen (lts)');          % Etiqueta del eje Y
        ylim([0 10])


        pause(0.05)  % Reduzco el tiempo de espera
        i=i+1;
        % Forzar la actualizaciï¿½n de la grï¿½fica
    end
else
    disp("No estas en modo automatico")
end
guidata(hObject, handles);


% hObject    handle to boton_pid (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in boton_fuzzy.
function boton_fuzzy_Callback(hObject, eventdata, handles)
% hObject    handle to boton_fuzzy (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



function kp_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to kp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of kp as text
%        str2double(get(hObject,'String')) returns contents of kp as a double
% --- Executes during object creation, after setting all properties.
function kp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to kp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function kd_Callback(hObject, eventdata, handles)
% hObject    handle to kd (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of kd as text
%        str2double(get(hObject,'String')) returns contents of kd as a double


% --- Executes during object creation, after setting all properties.
function kd_CreateFcn(hObject, eventdata, handles)
% hObject    handle to kd (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function ki_Callback(hObject, eventdata, handles)
% hObject    handle to ki (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of ki as text
%        str2double(get(hObject,'String')) returns contents of ki as a double


% --- Executes during object creation, after setting all properties.
function ki_CreateFcn(hObject, eventdata, handles)
% hObject    handle to ki (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in conectar.
function conectar_Callback(hObject, eventdata, handles)
try
    handles.mypi = raspi();
    set(handles.mensaje, 'String','Conexión exitosa con Raspberry Pi.');
    handles.conexion = 1;

    disp('Conexión exitosa con Raspberry Pi.');
    handles.continuarAnalisis = true;
```

```
    % Agrega aquí cualquier operación adicional que necesites realizar después de la conexión exitosa

catch
    set(handles.mensaje, 'String','Error al conectar con Raspberry Pi.');
    disp('Error al conectar con Raspberry Pi.');
    handles.conexion = 0;
end

% Configurar los pines para salida o entrada
configurePin(handles.mypi, handles.trigPin, 'DigitalOutput');
configurePin(handles.mypi, handles.echoPin, 'DigitalInput');
configurePin(handles.mypi, handles.rele, 'DigitalOutput');
configurePin(handles.mypi, handles.bomba_pwm, 'PWM'); % Crear un objeto
writePWMFrequency(handles.mypi, handles.bomba_pwm, 2000);
configurePin(handles.mypi, handles.rele, 'DigitalOutput');

% Inicializar vectores para almacenar datos
numMuestras = 5;  % Número de muestras que deseas recopilar
datosVolumen = zeros(1, numMuestras);
datosTiempo = datetime(zeros(1, numMuestras), 0, 0); % Inicializa un array de tipo datetime

pause(2)
set(handles.mensaje, 'String','Obteniendo el Volumen Actual');
% Recoger muestras de distancia
for i = 1:numMuestras
    datosVolumen(i) = measureDistance(handles.mypi, handles.trigPin, handles.echoPin);
    datosTiempo(i) = datetime('now');
    pause(1); % Pausa de 1 segundo entre mediciones
end

% Obtener el handle del axes
axesHandle1 = handles.axes1;  % Asegúrate de ajustar 'axes1' al Tag de tu axes
% Graficar en el axes
plot(axesHandle1,datosTiempo,datosVolumen);
% Configurar propiedades del axes
title(axesHandle1, 'Control de Nivel');  % Agrega un título
xlabel(axesHandle1, 'Tiempo');              % Etiqueta del eje X
ylabel(axesHandle1, 'Volumen (lts)');          % Etiqueta del eje Y
ylim([0 10])


guidata(hObject, handles);

% hObject    handle to conectar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes on button press in pushbutton11.
function pushbutton11_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% Función para medir la distancia
function volumen = measureDistance(mypi, trigPin, echoPin)
trigPin = 17;
echoPin = 27;

% Configurar los pines para salida o entrada
configurePin(mypi, trigPin, 'DigitalOutput');
configurePin(mypi, echoPin, 'DigitalInput');

volumen_buffer = [];       % Array vacío para el buffer de volumen

writeDigitalPin(mypi, trigPin, 0);
pause(0.5);
writeDigitalPin(mypi, trigPin, 1);
pause(0.00001);
writeDigitalPin(mypi, trigPin, 0);

% Medir el tiempo de respuesta
while readDigitalPin(mypi, echoPin) == 0
    break
```

```
end
inicio = tic;
while readDigitalPin(mypi, echoPin) == 1
    break
end
duracion = toc(inicio);

% Calcular la distancia
distancia = duracion * 34300 / 2; % Velocidad del sonido en el aire: 343 m/s
% Calcular la distancia
distancia2 = 23 - distancia;
if distancia2 <= 0
    distancia2 = 0;
end

volumen2 = (pi * (21.5)^2 * distancia2) / 1000;
x = volumen2;
volumen = 0.0023 * x^3 - 0.0668 * x^2 + 0.8638 * x + 0.2938;
if volumen <= 0
    volumen = 0;
end

% Filtro de media móvil
volumen_buffer = [volumen_buffer, volumen];
window_size = 10; % Ajusta esto según sea necesario
if length(volumen_buffer) > window_size
    volumen_buffer = volumen_buffer(2:end);
    volumen = mean(volumen_buffer);
end
```

BIBLIOGRAFÍA

Amllick, S. K. (2011). Study of the design and tuning methods of pid controller based on fuzzy logic and genetic algorithm.

Arbildo Lopez, A. (2011). El control de procesos industriales y su influencia en el mantenimiento. *Ingenieria Industrial*(29), 35-49. Recuperado el Noviembre de 2023

Astrom, K. (2000). *Control de procesos por computadora.* Mexico: Prentice Hall.

Bonilla-Fabela, I., Salazar, T., Arturo, M.-E., Guajardo Muñoz, M., & Luz Tania, I. &. (2015-2016). IOT, EL INTERNET DE LAS COSAS Y LA INNOVACIÓN DE SUS APLICACIONES. *Vinculategica, 2*(1). Obtenido de http://www.web.facpya.uanl.mx/Vinculategica/Revistas/R2/2313-2340%20-%20Iot,%20El%20Internet%20De%20Las%20Cosas%20Y%20La%20Innovacion%20De%20Sus%20Aplicaciones.pdf

Ibrahim, I. N., & Al Akkad, M. (2016). Exploiting an intelligent fuzzy-PID system in nonlinear aircraft pitch control. *International Siberian Conference on Control and Communications (SIBCON)*, (págs. 1-7). Moscow, Rusia. doi:10.1109/SIBCON.2016.7491828.

Katsuhiko, O. (2010). *Ingenieria de Control Moderna.* Madrid, España: Pearson Education.

Padhee, S. (2011). Control of a Ball and Beam System. Obtenido de http://data.mecheng.adelaide.edu.au/robotics/projects/2007/BallBeam/Wei_Final_Thesis.pd

Raspberry Ltda. (21 de JUNIO de 2019). *Raspberry Pi 4 Model B.* Obtenido de https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf

Raspberry Pi Ltda. (14 de Junio de 2023). *Raspberry Pi.* Recuperado el Noviembre de 2023, de https://datasheets.raspberrypi.com/rp2040/hardware-design-with-rp2040.pdf

Rose, K., Eldridge, S., & Chapin, L. (Octubre de 2015). *Internet Society.* Obtenido de https://www.internetsociety.org/wp-content/uploads/2017/09/report-InternetOfThings-20160817-es-1.pdf

Sisinni, E., Saifullah, A., Han, S., & Gidlund., U. J. (11 de Noviembre de 2018). *IEEE Transactions on Industrial Informatics.* doi:10.1109/TII.2018.2852491

Zadeh, L. A. (1988). *Fuzzy Logic.*