UNIVERSIDAD POLITÉCNICA SALESIANA
SEDE GUAYAQUIL

CARRERA DE INGENIERÍA EN ELECTRÓNICA

TEMA:

"IMPLEMENTACIÓN DE UN MÓDULO IoT HIDROPÓNICO NFT SEMIAUTOMÁTICO CON ALIMENTADOR DE NUTRIENTES POR CONTROL DIFUSO"

Trabajo de titulación previo a la obtencion del Título de Ingeniero Electrónico

AUTORES:

RECALDE DICADO KEVIN ARTURO
ROJAS MEZA EDUARDO XAVIER

TUTOR:

Ing. VICENTE PEÑARANDA MsC.

Guayaquil - Ecuador
2022

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros, **Kevin Arturo Recalde Dicado** con documento de identificación N° **0803017078** y **Eduardo Xavier Rojas Meza** con documento de identificación N° **0919557694** ; manifestamos que:

Somos los autores y responsables del presente trabajo; y,autorizamos a que sin fines de lucro la **Universidad Politécnica Salesiana** pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Guayaquil, 16 de Septiembre del año 2022

Atentamente,

Kevin Arturo Recalde Dicado	**Eduardo Xavier Rojas Meza**
CI: 0803017078	CI: 0919557694

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Nosotros, **Kevin Arturo Recalde Dicado** con documento de identificación N° **0803017078** y **Eduardo Xavier Rojas Meza** con documento de identificación Nº **0919557694**, expresamos nuestra voluntad y por medio del presente documento cedemos a la **Universidad Politécnica Salesiana** la titularidad sobre los derechos patrimoniales en virtud de que somos autores del trabajo de grado titulado: **"IMPLEMENTACIÓN DE UN MÓDULO IoT HIDROPÓNICO NFT SEMIAUTOMÁTICO CON ALIMENTADOR DE NUTRIENTES POR CONTROL DIFUSO",** el cual ha sido desarrollado para optar por el título de **Ingeniero Electrónico**, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribimos este documento en el momento que hacemos la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana

Guayaquil, 16 de septiembre de 2022

Atentamente,

Kevin Arturo Recalde Dicado	**Eduardo Xavier Rojas Meza**
CI: 0803017078	CI: 0919557694

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, **Vicente Avelino Peñaranda Idrovo** con documento de identificación N°0916113426 , docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: **"IMPLEMENTACIÓN DE UN MÓDULO IoT HIDROPÓNICO NFT SEMIAUTOMÁTICO CON ALIMENTADOR DE NUTRIENTES POR CONTROL DIFUSO",** realizado por **Kevin Arturo Recalde Dicado** con documento de identificación N° **0803017078** y **Eduardo Xavier Rojas Meza** con documento de identificación Nº **0919557694**, obteniendo como resultado final el trabajo de titulación bajo opción de Trabajo de Grado que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Guayaquil, 16 de Septiembre de 2022

Atentamente,

Ing. Vicente Avelino Peñaranda Idrovo, Msc.
CI: 0916113426

DEDICATORIA I

El éxito de este trabajo debo dedicárselo a Dios en primer lugar por brindarnos la fuerza, la paciencia y la perseverancia que se logró para realizar este proyecto.

Es mi deseo también el poder reconocer a mis padres por todo el tiempo que me han brindado su apoyo y preocupación, casi al borde de empujarme a poder conseguirlo.

Dedico también este trabajo a mi asombrosa esposa y a mi maravillosa hija quienes siempre me han esperado con amor cada vez que tuve que estar ausente para la realización de este proyecto.

Y por último deseo dedicar este logro a mí mismo ya que a lo largo de este tiempo he aprendido que es necesario poder valorar y reconocer mi propio esfuerzo para así poder seguir adelante ante cada obstáculo.

Ing. Kevin Recalde

DEDICATORIA II

El presente trabajo investigativo lo dedico principalmente a Dios, por ser el inspirador y darme fuerza para continuar en este proceso de obtener uno de los anhelos más deseados.

A mis padres, por su amor, trabajo y sacrificio en todos estos años, a mi esposa e hijos , por ustedes he logrado llegar hasta aquí y convertirme en lo que hoy soy.

A todas las personas que me han apoyado y han hecho que el trabajo se realice con éxito en especial a aquellos que me abrieron las puertas y compartieron sus conocimientos.

Ing. Eduardo Rojas

AGRADECIMIENTOS I

La gratitud es detenerse a darse cuenta y valorar las cosas que solemos dar por sentadas. Es por eso en esta ocasión es un placer para mi poder agradecer a todos lo que hicieron posible este trabajo de titulación entre los cuales están:

Mi familia en general mis padres, hermanos, esposa e hija, mis suegros, y demás familia de mi esposa por cada palabra de aliento.

Gracias a mis buenos amigos quienes son también un gran motivo para progresar a través de sus ánimos.

Un agradecimiento tambien a aquellos a quienes no ha tocado acudir en busca de asesoramiento y guia, por su entrega e interes en ayudar y ver cumplida esta meta.

Gracias tambien a nuestro tutor por su tiempo entregado en este proyecto, por sus amables correcciones y su paciencia para impulsarnos a terminar con los objetivos planteados.

Y por ultimo y mas importante a Dios, por reunir a todas estas personas y oportunidades en mi vida para finalmente concluir este trabajo.

Ing. Kevin Recalde

AGRADECIMIENTOS II

Agradezco a Dios por bendecirnos la vida, por guiarnos a lo largo de mi existencia, ser el apoyo y fortaleza en aquellos momentos de dificultad y de debilidad.

A mi familia, por haberme dado la oportunidad de formarme en esta prestigiosa universidad y haber sido mi apoyo durante todo este tiempo.

De manera especial a mi tutor de tesis, por haberme guiado, no solo en la elaboración de este trabajo de titulación, sino a lo largo de mi carrera universitaria y haberme brindado el apoyo para desarrollarme profesionalmente y seguir cultivando mis valores.

Ing. Eduardo Rojas

RESUMEN

La agricultura tradicional hace referencia a cultivos en suelo, los cuales han sido los más conocidos desde el inicio de las civilizaciones del ser humano, sin embargo existe desde la antigüedad evidencia de otro tipo de cultivo que no requiere un sustrato de tipo sólido; este tipo de cultivos son conocidos como hidropónicos, consisten en plantas cuyas raíces están en contacto directo con agua recirculante que contiene los nutrientes fundamentales para su crecimiento, y son objeto principal de este proyecto de titulación. El huerto hidropónico NFT semiautomático consiste en una eficiente y novedosa herramienta modular para cultivos que reemplaza el uso de suelo y abono, por agua con nutrientes disueltos, es adaptable a todo tipo de espacios residenciales y puede ser controlado y monitoreado de manera remota a través de la internet. La portabilidad del módulo facilita que sea instalado en patios o terrazas, y su adaptabilidad permite su funcionamiento en ecosistemas con climas extremos como páramos o desiertos. De esta forma representa una solución a problemas futuros tales como son la contaminación de suelos y falta de alimentos. La aplicación de tecnología IoT en el módulo utiliza el protocolo MQTT para comunicar el sistema con una aplicación a través de un broker que envía y recibe datos con un servidor en la nube, todo esto con soporte en plataformas de tarjetas embebidas como Raspberry Pi, con interfaces gráficas en sistema operativo Raspbian y otra en navegador web programada con diagramas de flujo en Node-RED. La escalabilidad con la que fue construido el módulo hidropónico permite a otros investigadores aplicar estas técnicas a sus trabajos, dado que el manejo de actuadores y control retroalimentado por medio de sensores permite una infinidad de aplicaciones en la actualidad. La investigación sobre

sistemas hidropónicos fue determinante en la selección del controlador para lo cual se eligió uno de lógica difusa el cual se encarga de la regulación de variables que influyen en la composición y características nutritivas del agua recirculante que ayuda en el crecimiento de las plantas.

Palabras Claves: Hidroponia, IoT, Control Difuso

ABSTRACT

Traditional agriculture refers to soil crops, which have been the best known since the beginning of human civilizations, however there has been evidence since ancient times of another type of crop that does not require a solid-type substrate; These types of crops are known as hydroponics, they consist of plants whose roots are in direct contact with recirculating water that contains the fundamental nutrients for their growth, and they are the main object of this investigation. The semi-automatic NFT hydroponic garden consists of an efficient and innovative modular tool for crops that replaces the use of soil and fertilizer with water with dissolved nutrients, it is adaptable to all types of residential spaces and can be controlled and monitored remotely through the Internet. The module's portability makes it easy to install on patios or terraces, and its adaptability allows it to function in ecosystems with extreme climates such as moors or deserts. In this way it represents a solution to future problems such as soil contamination and lack of food. The IoT technology application in the module uses the MQTT protocol to communicate the system with an application through a broker that sends and receives data with a server in the cloud, all this with support on embedded card platforms such as Raspberry Pi, with graphical interfaces in the Raspbian operating system and another in a web browser programmed with flowcharts in Node-RED. The scalability with which the hydroponic module was built allows other researchers to apply these techniques to their work, since the management of actuators and feedback control through sensors allows an infinity of applications today. Research on hydroponic systems was decisive in the selection of the controller, for which one of fuzzy logic was chosen, which is

responsible for the regulation of variables that influence the composition and nutritional characteristics of the recirculating water that helps in the growth of plants.

Key words: Hydroponics, IoT, Control Fuzzy

Tabla de Contenidos

Índice de Figuras

Índice de Tablas

INTRODUCCIÓN

La emergencia de salud a inicios del año 2020 tuvo como efecto colateral el colapso de la cadena de suministro alimenticia, de forma que ciertos productos agrícolas no llegaban a los mercados por los bloqueos entre cantones, mientras que otros se encarecieron por el bajo stock y la alta demanda.

Así mismo era notoria la carencia de espacios apropiados donde poder producir este tipo de alimentos dentro de la ciudad, dado que un factor determinante en cultivos típicos es un suelo con los nutrientes ideales, oxigenación, expuesto a sol a determinadas horas del día y poco vulnerable a inundaciones y otras inclemencias de la naturaleza.

Como estudiantes de la Universidad Politécnica Salesiana, además de la formación académica, educación en valores y ética, por lo que este problema llamó nuestra atención. Es por eso que se decidio tomar esta problemática y convertirlo en una oportunidad para brindar una solucion.

En el primer capítulo trata el problema y todas sus implicaciones, explicando los objetivos y delimitando el alcance del proyecto. El capítulo dos contiene la sustentación teórica de esta investigación, con las citas correspondientes que se podrán revisar al final del documento. Un tercer capítulo se detalla el desarrollo del sistema y su implementación como módulo. Y finalmente en el cuarto capítulo tiene las conclusiones y recomendaciones.

CAPÍTULO I

EL PROBLEMA

1.1. Tema

Implementación de un módulo IoT hidropónico NFT semiautomático con alimentador de nutrientes por control difuso.

1.2. Planteamiento del Problema

Un evento trascendental en la sociedad ocurrió a inicios del año 2020, donde las zonas metropolitanas evidenciaron el colapso de la cadena de suministro alimenticia, afectando al abastecimiento y dificultando así el acceso a víveres por medios comerciales. Por lo que aquellos con acceso a terrenos cultivables en sus hogares, iniciaron huertos familiares para suplir necesidades dentro de sus posibilidades. Pero ¿Qué pasa cuando no se cuenta con espacios para cultivos tradicionales? La implementación de cultivos que no dependan de acceso a terrenos fértiles se vuelve imperativo para conseguir una economía circular que sostenga tanto a las familias y a la sociedad, como al ecosistema en el que se desarrollan.

1.3. Importancia y Alcance

Un cultivo tradicional no sólo se enfrenta a la problemática que representa la carencia de espacio físico, sino también a la degradación de los nutrientes que éste contiene, lo que hace que con el tiempo necesiten readecuación mediante aplicación de abonos, o la rotación de cultivos por temporadas para dar paso a la regeneración de nutrientes utilizados por un tipo específico de planta. Frente a esto gana importancia la implementación de cultivos que permitan la regeneración de estos suelos, al tiempo que ayudan a disminuir el uso de fertilizantes evitando la

contaminación del agua con los químicos que estos contienen. Los huertos hidropónicos presentan las características señaladas, convirtiéndose así en parte fundamental de una economía circular enfocada en la mejora de la producción mediante la preservación y/o uso eficiente de recursos naturales renovables y no renovables.

1.4. Delimitación

1.4.1. Temporal

La implementación de este proyecto se realizó en el periodo lectivo número 58, hasta la fecha actual.

1.4.2. Espacial

El proyecto fue desarrollado en el domicilio de uno de los autores, ubicado en las calles Andrés Marín y Alcedo, del barrio Garay, en la ciudad de Guayaquil, debido a que el acceso a las instalaciones de la Universidad Politécnica Salesiana Sede Guayaquil se encontraba restringido a causa de la pandemia por Coronavirus.

1.4.3. Académica

El proyecto “Implementación de un módulo IoT hidropónico NFT semiautomático con alimentador de nutrientes por control difuso” consiste en una herramienta modular para cultivos sin necesidad de suelo, adaptable a espacios residenciales y que puede ser monitoreado y controlado de manera remota. En este proyecto se aplicaron conocimientos adquiridos en las materias: Teoría de Control II y III, Redes de Computadoras III, Sensores y Transductores e Informática Industrial.

1.5. Objetivos

1.5.1. Objetivo General

Implementar un módulo IoT hidropónico NFT semiautomático con alimentador de nutrientes por control difuso ser ubicado en la terraza del edificio de biotecnología en la Universidad Politécnica Salesiana Sede Guayaquil.

1.5.2. Objetivos Específicos

- Diseñar e implementar un sistema hidropónico NFT modular para cultivar huertos hidropónicos.
- Conectar el módulo hidropónico mediante un sistema IoT amigable con el usuario.
- Implementar un control difuso de dosificación de nutrientes relativo a la demanda del cultivo.
- Cultivar y cosechar una variedad de lechuga crespa o Lactuta sativa var. Crispa por su nombre científico en el sistema hidropónico.
- Conectar el módulo a un sistema de notificaciones redundante por tecnología WiFi y GSM.
- Elaborar un libro con 4 prácticas dirigidas a la carrera de Biotecnología para que sus estudiantes puedan operar el sistema guiados por docentes de la Universidad Politécnica Salesiana.

CAPÍTULO II

FUNDAMENTOS TEÓRICOS

2.1. Hidroponia

2.1.1. Concepto

Hidroponía es una palabra compuesta por dos términos griegos, *Hidro*, que significa “agua” y *Ponos*, que se usa para referirse a trabajo o labor. En traducción textual se definiría a hidroponía como la labor que se hace en el agua, sin embargo, en la actualidad se usa este término para describir a los cultivos que se desarrollan sin requerimiento de suelos, así como al conjunto de técnicas que facilitan este fin.

Históricamente la primera civilización que desarrolló estos cultivos fueron los Babilonios en el siglo VI a.C., con la creación de la que es considerada una de las Siete Maravillas del Mundo Antiguo, conocida como los Jardines Colgantes de Babilonia y que se aprecian en la Figura 1. A pesar que no se conoce con precisión su funcionamiento, se sabe por historiadores que una noria llevaba agua hasta los sitios más altos de la edificación, la cual luego fluía por los diversos niveles de la estructura (Tendenzias, 2019).

Figura 1: Jardines Colgantes de Babilonia (Noguès, 2021).

Mediante la hidroponía es posible cosechar hortalizas de óptima calidad, promueve el uso eficiente de recursos como son el agua y los nutrientes, además del espacio físico, dado que no requiere suelos fértiles. La técnica en la que la raíz está en contacto únicamente con el agua y toma sus nutrientes de ella, es conocida como Nutirent Film Technique (NFT). En esta técnica es requerido que el agua sea aireada por medios físicos (Beltrano, 2015).

2.1.2. Agua

El agua juega un papel primordial en la alimentación de la planta, ya que esta lleva los nutrientes hasta cada célula vegetal que la conforman. En los cultivos tradicionales las raíces crecen más ya que necesitan incrementar el área de contacto con el suelo para que pase suficiente agua hacia la planta, en los cultivos hidropónicos es común que las raíces sean menos desarrolladas, ya que están en contacto directo con el agua y las soluciones nutritivas que contiene.

El balance hídrico en una planta depende de tres procesos básicos, la absorción a través de las raíces, la conducción a través del xilema y la transpiración, que es la liberación de agua en forma de vapor a través de los estomas. El nivel de agua suele cambiar a lo largo del día y la noche, dependiendo de estos factores y de la especie de planta. Se conoce que un déficit hídrico en cultivos hidropónicos es muy poco frecuente, pues las raíces se encuentran sumergidas en agua. Esto es ventajoso, pues un déficit hídrico puede afectar directamente a la producción (Ruscitti, 2015).

2.1.3. Fotosíntesis

Es el proceso químico mediante el cual la planta transforma el dióxido de carbono (CO_2) en glucosa ($C_6H_{12}O_6$), en esta reacción interviene además el agua y la energía solar, y se conoce comúnmente que es el proceso inverso a la respiración que realizan los mamíferos. En la Figura 2 se encuentra la fórmula química de la fotosíntesis.

$$6\ CO_2 + 12\ H_2O + \text{Energía (solar)} \longrightarrow C_6H_{12}O_6 + 6\ O_2 + 6\ H_2O$$

Figura 2: Fórmula química de la fotosíntes (Giménez, 2015)

En cultivos hidropónicos de invernadero se utilizan métodos en los que se incrementa el contenido de CO2 en el ambiente controlado del cultivo. A este proceso se conoce como enriquecimiento de CO2, y ha demostrado mejorar la fotosíntesis si se aplica en la medida adecuada en las etapas de producción de flores y frutos. Esta intermitencia en el enriquecimiento se empezó a aplicar cuando se notó que la planta se aclimata al aumento de CO2, de manera que su producción incrementaba al principio, pero luego se normalizaba (GroHo, 2021).

2.1.4. Soluciones Nutritivas

Las soluciones nutritivas son la mezcla de agua y sales minerales disociadas en alta concentración que se agrega al agua circulante de un sistema hidropónico y permite el crecimiento de las plantas al intercambiar nutrientes por minerales de desecho con sus raíces. En ese sentido, se pueden mencionar algunos parámetros de gran importancia en una solución nutritiva óptima.

La oxigenación del agua debe mantenerse de lo contrario disminuye la absorción del hierro (Fe), lo que genera el aprovechamiento deficiente de iones potasio (K) y nitrato (NO3) en la planta, nutrientes esenciales para el desarrollo de sus raíces. Respecto al contenido de minerales se nota el antagonismo del potasio (K) y de iones fosfato (PO4), que en altas concentraciones dificultan la absorción del magnesio (Mg) y el hierro respectivamente, por lo que se debe cuidar que su contenido no se encuentre elevado (Carbone, 2015).

Existen manuales que indican las proporciones en las que se puede combinar los distintos compuestos para elaborar una solución nutritiva afín a las necesidades del cultivo, sin embargo se decidió utilizar una de las alternativas de soluciones nutritivas del mercado con mejores respuestas positivas de los consumidores. De esta manera se enfoca este proyecto, en la medición principal de la electroconductividad de la solución nutritiva por medio del sensor de TDS y su monitoreo remoto mediante el módulo IoT.

2.1.5. Clima y Acondicionamiento

En cultivos tradicionales las condiciones óptimas para el cultivo de lechuga se dan a una altura entre los 1800 a 2800 msnm, en suelos semiarcillosos de pH entre 5.2 y 6.7. Durante el día se prefiere la temperatura de 15 a 18 ºC, para conseguir una cantidad mínima de tallos florales. Temperaturas extremas, 7 ºC para el mínimo y 21 ºC máximo, facilitan la producción prematura de tallos, lo que baja la calidad del cultivo (Rogel, 2018).

Los cultivos hidropónicos implementados a la altura del nivel del mar son viables siempre que el clima sea templado. Los valores óptimos de pH son de 5.0 a 5.5, de

temperatura son de 16 a 22 ºC, con separación de 30 cm entre plantas. En estas condiciones el ciclo vegetal de la lechuga toma de 90 a 100 días (Urdiales & Espín, 2018).

2.1.6. Estructura NFT (Nutrient Film Technique)

Es la técnica de la solución nutritiva recirculante, y consiste en la disolución de los nutrientes en el agua que recorre por el sistema y que al estrechar contacto con las raíces las provee de alimento. Para esta técnica se vuelve necesaria la oxigenación del agua, por lo que en algún momento debe estar en contacto con el aire; así se puede evitar la acumulación de CO2 en las raíces (Andreau, 2015).

2.2. Electrónica

2.2.1. Válvulas Solenoides

Son dispositivos que abren o cierran el paso de fluidos (gaseosos o líquidos) en respuesta al impulso magnético de un electroimán, que mueve un muelle y lo devuelve a su posición al activarse y desactivarse. Estas válvulas de paso no pueden regular el flujo, sino únicamente abrir o cerrar el paso (Sanitron, 2019).

La válvula utilizada para esta implementación es una válvula de dos vías.

2.2.2. Bombas Peristálticas

Este sistema de bombeo consiste en la rotación de un sistema con rodillos que se alternan para presionar una manguera y así aprovechar la presión del vacío en el siguiente paso del rodillo. Del mismo modo al estar ocluida la manguera se crea una acción de desplazamiento positivo que evita que exista flujo en dirección contraria cuando la bomba no se encuentra encendida (Watson Marlow, 2022).

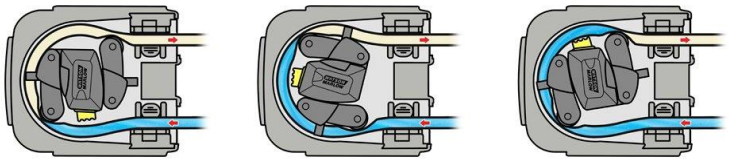


Figura 3: Funcionamiento de una bomba peristáltica (Watson Marlow, 2022)

En la primera vuelta actúa la presión de la manguera por un lado y su alivio por el otro, lo que tira el fluido en la dirección deseada. Al girar, un segundo rodillo comprime la manguera, creando un sello que asegura que el flujo no vaya en dirección contraria. Al continuar el giro de los rodillos la presión en la manguera se alivia, lo que crea un vacío que permite la continuidad del flujo. Un diagrama del funcionamiento se puede observar en la Figura 3.

Entre las ventajas de las bombas peristálticas se menciona la inocuidad con la que se trata a los fluidos, ya que la bomba no forma la vía por donde este circula, sino que manipula la vía para permitir el paso. En este sentido, el único material de desgaste es la manguera, lo que hace que el requerimiento de mantenimiento de las bombas sea mínimo. Las bombas utilizadas en el proyecto tienen motor DC que se activan con 12V, y fueron conectadas a un driver IBT2 43A, para tener la disponibilidad de funcionamiento en sentido horario y antihorario.

2.2.1. Control de Motor de Alta Potencia IBT2 - BTS7960

El IBT2 es un controlador diseñado para solenoides y motores de corriente continua en un que funcionan en un amplio rango de potencia. Se puede utilizar tanto con microcontroladores como con tarjetas electrónicas embebidas. Su

funcionamiento es posible gracias a que contiene un MOSFET de lado alto en canal positivo y un MOSFET de lado bajo en canal negativo (MovilTronics, 2021).

2.2.2. Sensor de Temperatura DS18B20

Es un sensor digital para medición de temperatura en grados centígrados de 9 a 12 bit, con alarma de temperatura alta y baja, y alimentación en la línea de datos del cable bus. Su arquitectura permite conectar varios sensores de este tipo con el mismo cable, lo cual vuelve viable el uso de un microprocesador para controlar varios sensores destinados a un mismo proceso (Maxim Integrated, 2019). En la Figura 4 se puede apreciar al transductor en su forma más comercial, y entre las principales aplicaciones se mencionan sistemas industriales con control termostático y procesos con sistemas sensibles a la temperatura.

Figura 4: Sensor de temperatura digital DS18B20 (Naylamp, 2019)

2.2.3. Sensor de Presión Barométrica BMP280

Es un sensor de presión barométrica absoluta, de alta precisión y bajo consumo energético. Adicionalmente este sensor mide también la temperatura, la altitud y la humedad relativa. Sus aplicaciones más comunes son mejoras de la navegación

con Sistema de Posicionamiento Global (GPS), detección de altitud para elevadores y vehículos aéreos no tripulados, pronóstico del clima, espirometría (medición de la capacidad pulmonar) y medición de velocidad vertical (Bosch, 2015).

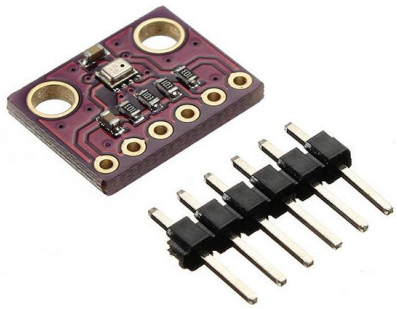


Figura 5: Tarjeta con sensor BMP280

2.2.4. Transmisor de Celda de Carga HX711

El trasmisor de celda de carga HX711 es un convertidor analógico a digital (ADC) de 24 bits, diseñado para brindar alta precisión en sus aplicaciones, que pueden ir desde balanzas digitales hasta aplicaciones de control industrial. El amplificador de ganancia programable puede funcionar en 32, 64 y 128. Su circuito power-on-reset simplifica la inicialización del componente (AVIA, 2020).

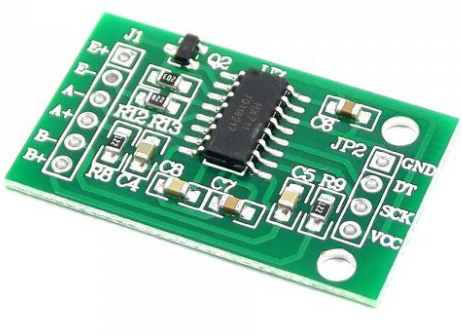


Figura 6: Módulo Controlador Sensor de Peso XFW-HX711 (Unitronic, 2020)

2.2.5. Sensor de Electroconductividad TDS

Es utilizado como un sensor de calidad de agua dado que mide las partículas disueltas en una solución. Entre estas partículas pueden mencionarse sales, minerales o metales. Su funcionamiento es posible gracias a un sensor de electroconductividad (EC) que detecta la cantidad de sólidos disueltos en el agua.

2.2.6. ESP32 GSM

Es una tarjeta de circuito impreso o PCB diseñada como un sistema embebido que combina las tecnologías wi-fi, bluetooth y GSM. Para su programación consta de un puerto USB-C, el cual también sirve para energizarla. La capacidad que tiene para recibir comandos vía mensaje de texto o llamada telefónica la hace una gran alternativa al momento de implementar desarrollos de IoT (Robotics ORG, 2021).

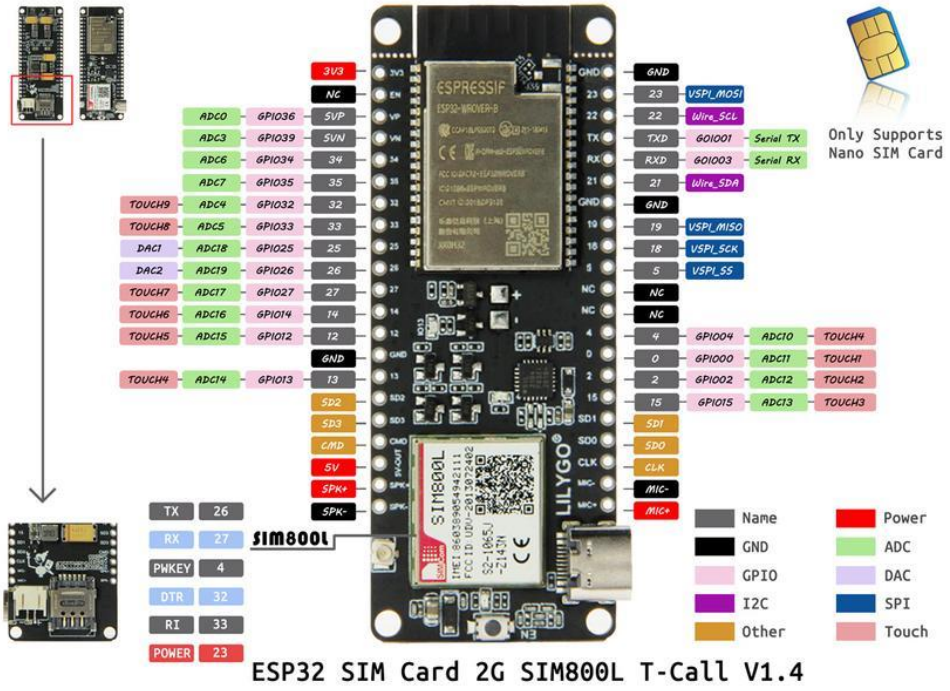


Figura 7: ESP32 GSM (Robotics ORG, 2021)

2.2.7. Raspberry Pi 4

La organización sin fines de lucro Raspberry Pi se encarga del desarrollo de tarjetas electrónicas para sistemas embebidos. La Raspberry Pi 4 fue sacada al mercado en el 2019. Tiene un procesador Broadcom BCM 2711 de 64 bits que consta de 4 núcleos con una velocidad de procesamiento de 1.5 GHz. Cuenta con tecnología WLAN y Bluetooth integrado. Además tiene un puerto Ethernet y GPIO de 40 pines, 4 puertos USB, salida estéreo y puerto HDMI (Raspberry Pi, 2021).

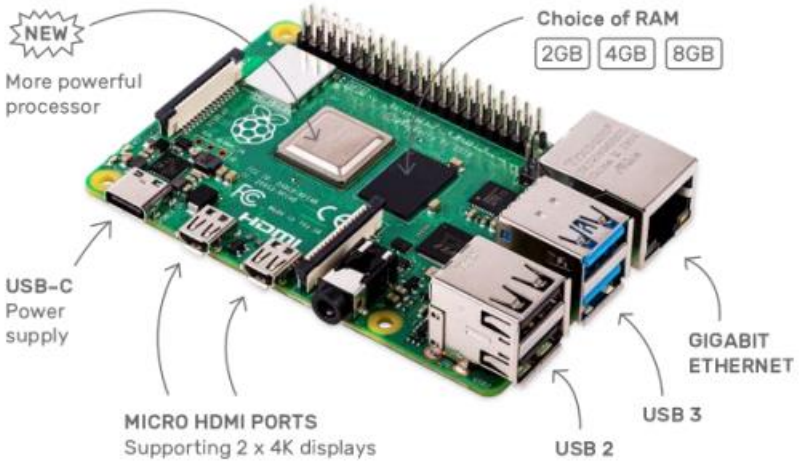


Figura 8: Raspberry Pi 4 Modelo B (Raspberry Pi, 2021)

2.3. Control Difuso

El control difuso originalmente traducción del inglés *"control fuzzy"* es utilizado para la obtención de un modelo a través de aproximaciones, en procesos del mundo real que son difíciles de definir con alta precisión. Estos sistemas de control son basados en reglas que utilizan las sentencias *"If / then"* para formar la base de conocimiento que defina el nivel de pertenencia de las diferentes funciones de membresía del proceso a controlar (Mosquera, 2014).

2.3.1. Conjunto Difuso

A diferencia de los conjuntos tradicionales en los que un elemento puede pertenecer en su totalidad a uno o varios conjuntos, en los conjuntos difusos ocurre que, ante la declaración de variables lingüísticas, se establece una medida entre cero (0) y uno (1), donde un elemento puede pertenecer en distinto grado a uno o más conjuntos (Hurtado, 2014).

En un ejemplo dado, como el conjunto de personas altas, bajo la lógica de conjuntos tradicionales se menciona la existencia de dos grupos: personas altas y no altas. En la lógica difusa, gracias a las funciones de membresía, es posible describir a un mismo sujeto como un poco alto o un poco bajo. La diferencia entre conjuntos clásicos y difusos se muestra en la Figura 9.

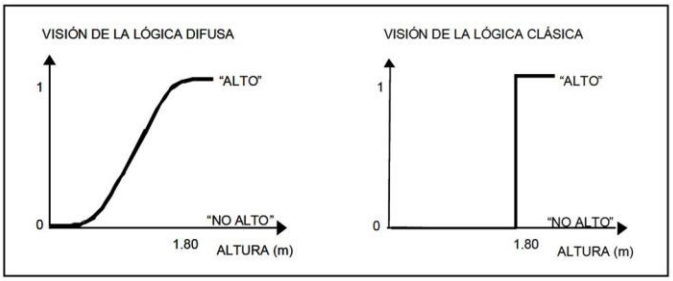


Figura 9: Diferencia entre conjuntos clásicos y difusos (Castro, 2016).

2.3.2. Universo de Discurso y Funciones de Membresía

Función de membresía son todos los elementos de un conjunto que pertenecen a un universo dado y que son descritas con una variable lingüística. Las más utilizadas son de tipo triangular y trapezoidal. Todos los valores finitos que pueden ser asignados a las variables lingüísticas toman el nombre de Universo de Discurso.

En lógica difusa un valor del Universo de discurso puede pertenecer en distinta medida a dos variables lingüísticas (Hurtado, 2014).

2.3.3. Estructura de un Controlador con Lógica Difusa

Los sistemas difusos funcionan con un conjunto de reglas que conforman la base de conocimiento, estas son sentencias condicionales de la forma "Si *x* es *A* entonces *y* es *B*" donde "*x"* y "*y"* son variables de entrada y salida respectivamente, y "*A"* y "*B"* son las diferentes funciones de membresía que manifiestan las variables. El proceso de control se observa en la Figura 10. "u" es una variable de entrada cuyos valores numéricos necesitan convertirse en una frase que pueda ser interpretada, a esta frase se llama variable lingüística, este proceso se llama *Fuzzyficación*.

Luego las variables lingüísticas ingresan al sistema de inferencia donde se realizan los cálculos matemáticos difusos de acuerdo a las reglas de inferencia establecidas por el programador. Los resultados o variables lingüísticas de salida ingresan luego al proceso llamado *"defuzzyficación"* que consiste en convertir nuevamente las variables lingüísticas, esta vez de las funciones de membresía de salida, en valores numéricos que retroalimentan el sistema para realizar la corrección del error.

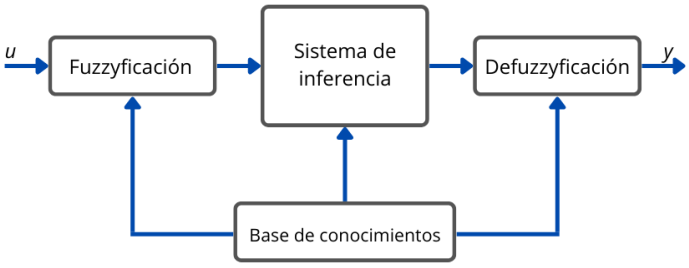


Figura 10: Estructura de un controlador difuso (Mashhad, 2013)

2.3.4. Mecanismo de Inferencia: Sistemas tipo Mamdani

Los principales mecanismos de inferencia son de tipo Mamdani y los de Sugeno. El controlador fuzzy diseñado para este proyecto corresponde al sistema Mamdani por lo que en este apartado se procede a describirlo. Los sistemas difusos de tipo Mamdani consisten en tres elementos principales: difusor, mecanismo de inferencia difusa, base de reglas difusas y desdifusor (Llano, 2007)

En la Figura 11 se explica el funcionamiento de este sistema. Cuando un valor puntual ingresa al sistema difuso interpreta un nivel de pertenencia en cada regla del sistema, donde se aplican los operadores difusos correspondientes para valores de pertenencia asociados a las entradas. Este resultado y su respectiva función de pertenencia de la salida asociada a la regla, ingresan al método de implicación, donde se aplica el operador difuso correspondiente. Luego, mediante otro operador difuso, se agregan la totalidad de las implicaciones, y al final se obtiene el resultado aplicando el método de defusificación.

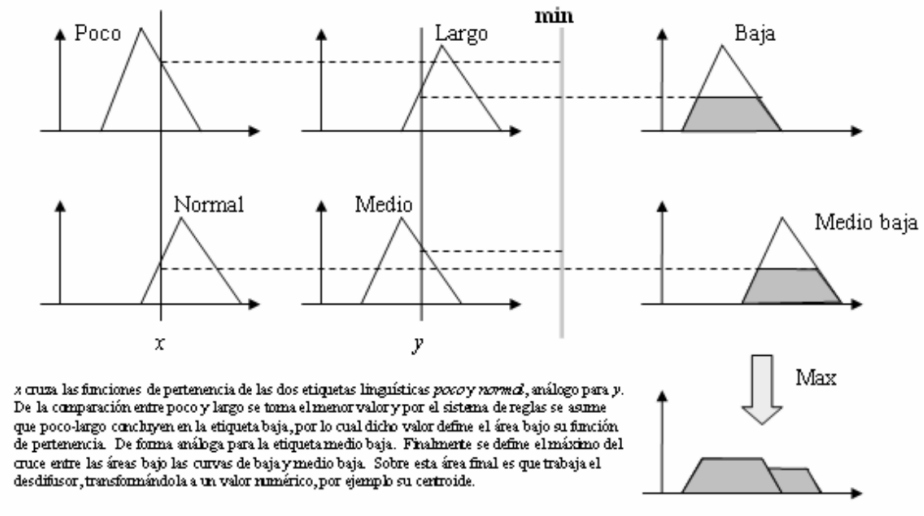


Figura 11: Funcionamiento de un controlador de tipo Mandani (Llano, 2007)

2.4. Sistema de Comunicación

2.4.1. Node Red

Node-Red es una herramienta de programación para vincular dispositivos electrónicos, interfaces de programación de aplicaciones y servicios en línea. Basado en Node.js utiliza lógica de diagramas de flujo, los que son programados a través de una ventana de navegador (Node-RED, 2022). Para este proyecto se instaló Node-RED en la Raspberry Pi y se asignó una IP para ejecutar el editor en el navegador de una portátil.

2.4.2. Protocolo MQTT

MQTT o Message Queing Telemetry Transport es un protocolo de comunicación Machine to Machine basado en TCP/IP y funciona como un servicio de mensajería push con patrón publicador/suscriptor. Esto significa que un cliente puede publicar un mensaje, el cual se organiza de manera jerárquica en tópicos, y otros clientes pueden suscribirse a este tópico para tener acceso al mensaje (Llamas, 2022).

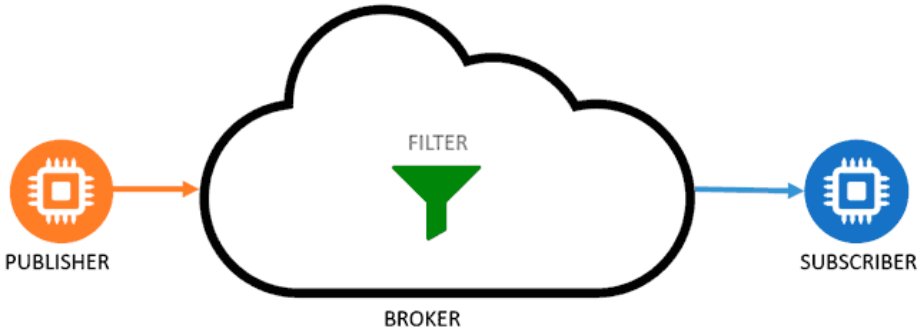


Figura 12: Topología de funcionamiento del protocolo MQTT (Llamas, 2022)

2.5. Internet of Things (IoT)

Este concepto propone entender la Internet como un paradigma que se extiende al mundo real, alcanzando objetos, data y software inteligente, e incluso seres humanos. Todos los mencionados anteriormente son reconocidos como actores de la infraestructura de la red, y como propósito, IoT se concentra en la creación de un entorno en el que estos actores puedan compartir información al instante, desde cualquier parte del mundo (Yang, 2013).

Con IoT es posible el monitoreo y control remoto de los huertos, con observabilidad en tiempo real de parámetros de humedad, temperatura, nivel de agua, nivel de PH, y manejo de variables en las recetas de solución nutritiva dependiendo de las necesidades que se detecten en las plantas (Ravi & Mohamed, 2020).

CAPÍTULO III

MARCO METODOLÓGICO

La implementación de proyectos de electrónica orientados a la solución de problemáticas de otras ciencias que conlleva a ampliar el conocimiento, en este proyecto se puede ver la versatilidad de los cultivos NFT y la facilidad que el flujo de nutrientes y oxígeno brinda al crecimiento de las plantas. Ya llevando a cabo la parte práctica, se encontraron obstáculos que requirió la modificación del producto final, sin embargo, se procuró buscar el cumplimiento de objetivos enfocando la parte de desarrollo electrónico sobre la botánica.

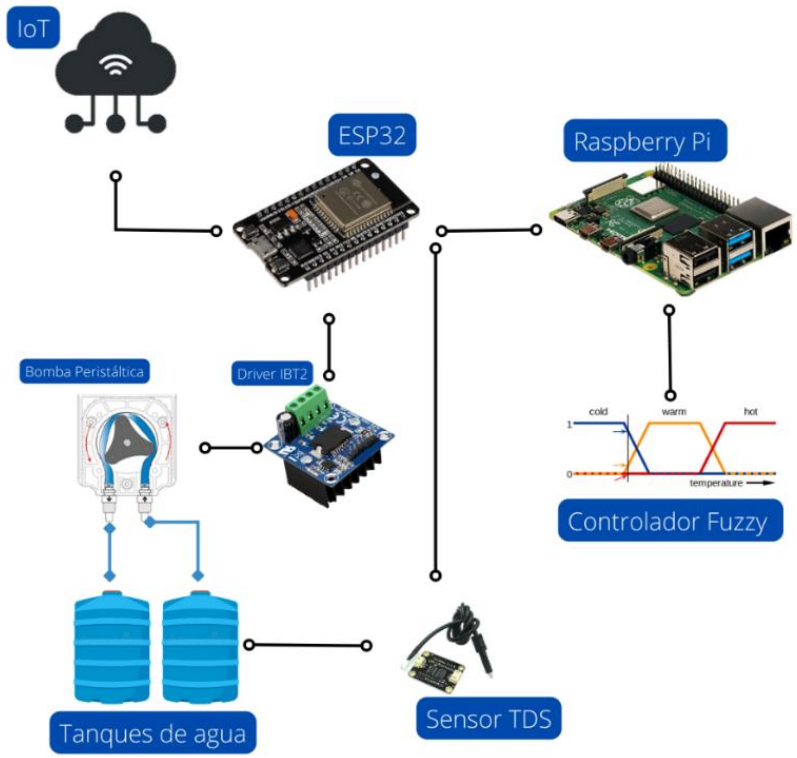


Figura 13: Esquema de Proyecto

Fuente: Elaborado por los Autores.

La combinación de tecnologías IoT a través del protocolo MQTT, los controladores fuzzy, el diseño electrónico de PCB junto a aplicaciones con tarjetas embebidas, hacen que sea posible generar un modelo de control para huertos hidropónicos novedoso que es objetivo principal de este proyecto de tesis. Por cuanto se lo realizó con escalabilidad no se descarta pueda ser utilizado por futuros estudiantes de pregrado o posgrado para proyectos de similar aplicación.

La importancia del sensor de pH radica en la necesidad de mantener un nivel óptimo de alcalinidad para que se puedan activar los mecanismos de nutrición y crecimiento de la planta, mientras que el sensor de conductividad brinda una retroalimentación de valores de concentración de nutrientes, para que los minerales requeridos por la planta para su crecimiento no falten o excedan los valores adecuados. Características de los sensores se describen en la

Tabla 1.

	Rango del sensor	**Rango ideal**
Sensor de pH	1 - 14	5.2 - 6.2
Sensor de EC	0 - 1000 PPM	500 - 750 PPM

Tabla 1: Valores de dominio y rango de los sensores

Fuente: Los autores

El rango del sensor son aquellos valores en los que el instrumento es capaz de realizar la medición, y fueron consultados en las hojas de datos de cada componente. El sensor de pH mide el potencial de hidrogeniones en magnitudes adimensionales en valores entre 1 y 14. El sensor de electroconductividad puede dar la concentración salina en soluciones entre 0 y 1000 PPM. En la columna rango ideal se definen los valores más adecuados para el sistema hidropónico NFT.

3.1. El Huerto

3.1.1. Estructura del huerto NFT

El prototipo fue montado en una estructura de repisas con tres niveles. El nivel superior tiene dos cañerías donde se colocan las plantas y el agua con nutrientes circula para su crecimiento. El nivel medio consta del panel de la planta y el panel del controlador. El nivel inferior tiene tres contenedores para el agua potable, la solución nutritiva y el tanque de almacenamiento de agua recirculante.

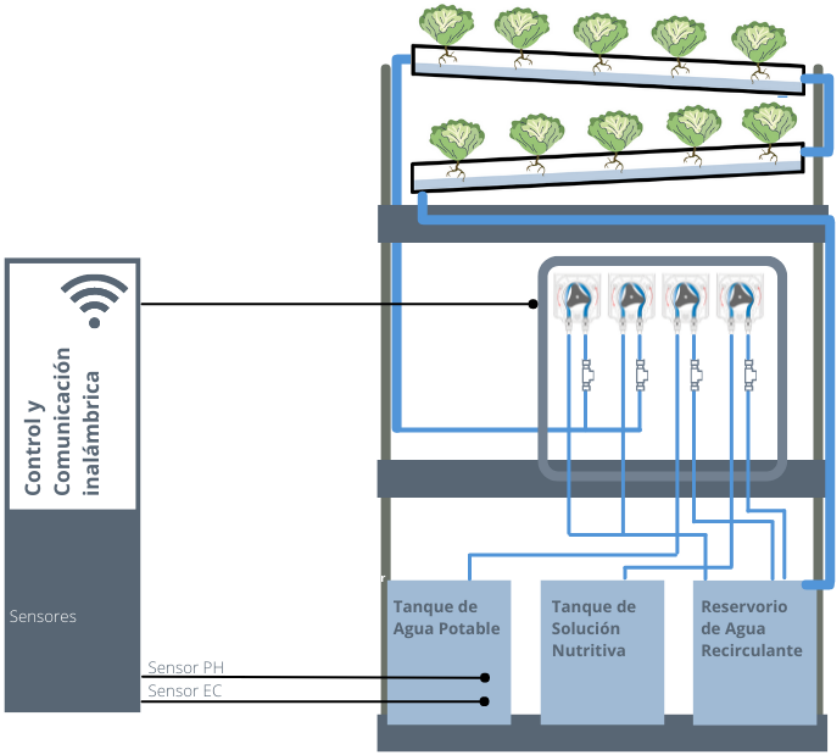


Figura 14: Diseño de Huerto NFT

Fuente: Elaborado por los Autores.

Se puede observar el panel de la planta en la Figura 16. En la puerta posee cuatro bombas peristálticas, una de ellas se encarga de la recirculación del agua, otra

mantiene el nivel drenando el exceso, una tercera bomba es suministro de solución nutritiva, y la última agrega agua potable, siendo estas dos últimas las encargadas de nivelar la concentración de solutos del agua que circula por las cañerías.

En la Figura 15 se observa en forma esquemática el orden de las bombas peristálticas tal como están ubicadas en la puerta del tablero de planta y los contenedores de líquidos rotulados, además, se observa en las flechas la dirección que llevan los fluidos a través del sistema.

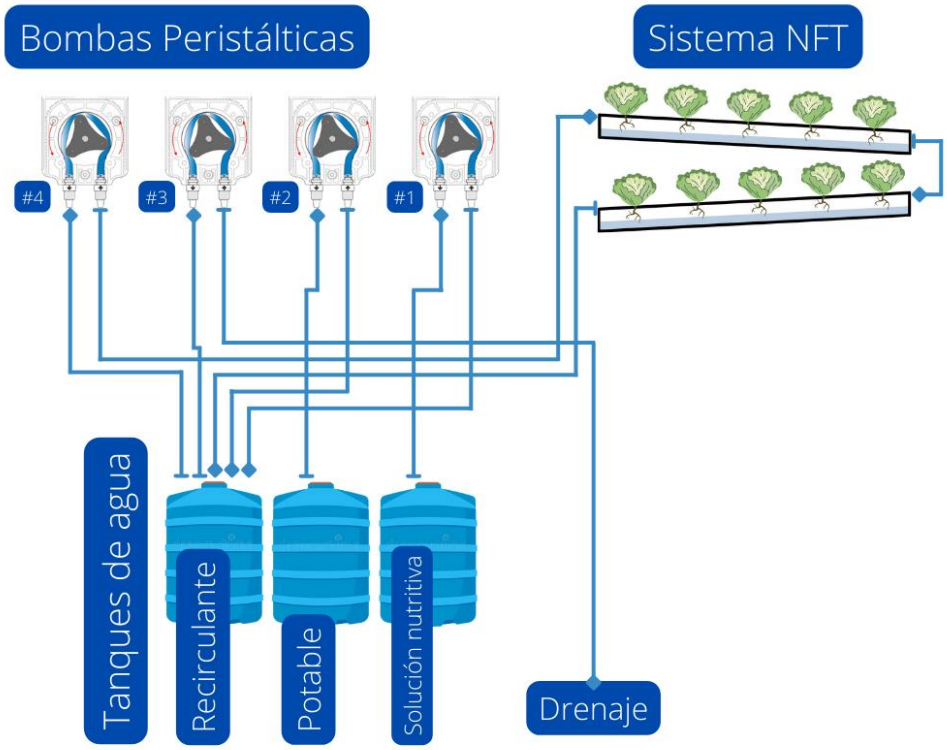


Figura 15: Esquema de bombas peristálticas y circulación de fluidos

Fuente: Elaborado por los Autores.

En el interior del panel de la planta se puede hallar las fuentes de poder que energizan todo el sistema. El módulo IoT programado en una tarjeta ESP32, en este

módulo se programó además de la redundancia GSM que consiste en líneas de código que comprueban la disponibilidad de red wi-fi. Cuatro módulos driver IBT2 de 43 amperios para el control de las bombas peristálticas. Dos sensores ambientales de temperatura y presión barométrica Bosch BMP280. Y el sensor de sales disueltas TDS que va al tanque de agua recirculante para recoger el valor de electroconductividad.

Figura 16: Montaje del tablero de la planta

Fuente: Elaborado por los Autores.

El panel del controlador tiene la Raspberry Pi donde se programó el controlador difuso en lenguaje Python. Esta tarjeta embebida además contiene el diagrama de flujo diseñado en Node-Red para el intercambio de información entre el módulo y una plataforma web alojada en una IP local, la cual permite el control y monitoreo del sistema IoT. A esta aplicación es posible acceder desde cualquier computadora gracias a que se configuró la IP para enviar su contenido a la nube.

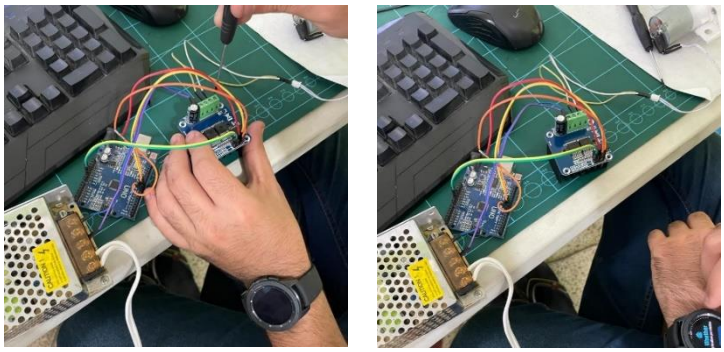


Figura 17: Prueba de funcionamiento de bombas peristálticas con driver de motor IBT2

Fuente: Elaborado por los Autores.

Los contenedores utilizados en el sistema están hechos de polipropileno, y sus medidas son 40 x 25 x 25 cm. Uno de los contenedores de líquidos fue llenado con agua potable y otro con la solución salina con concentración de 1000 PPM, ambos tienen dos orificios en la tapa, uno para la manguera que va a la bomba en el panel de control y la otra para el ingreso del sensor respectivo.

Figura 18: Tanques de almacenamiento de agua recirculante, potable y solución nutritiva

Fuente: Elaborado por los Autores.

El tercer contenedor es el de agua recirculante y contiene la mezcla de agua y solución nutritiva que va a fluir constantemente por las cañerías para nutrición y

crecimiento de las plantas; consta de cuatro agujeros, de los cuales dos son para los sensores y dos para las mangueras que vinen de las bombas del panel de planta. Adicional a esto se incorporó una boya de nivel, que al activarse inicia el ciclo de recirculación gracias a la activación de dos de las bombas peristálticas en el panel de control. Los contenedores descritos previamente se observan en la Figura 18.

3.1.2. Solución Nutritiva

Para la solución nutritiva existen opciones comerciales que se pueden adquirir para mezclar y usar, aunque también es posible mezclar por cuenta propia los nutrientes en la cantidad que más se ajuste a las necesidades. En la

Tabla 2 están anotadas las cantidades en gramos para la elaboración de dos tipos de solución nutritiva. En ambos casos, se debe disolver las cantidades justas en 200 litros de agua.

Tabla 2: Composición mineral de dos tipos de soluciones nutritivas

	Solución nutritiva de Bechhart y Connors	**Solución nutritiva de Universidad de California**
Nitrato de calcio	486 g	90 g
Nitrato de potasio	-	90 g
Fosfato monopotásico	57 g	-
Fosfato monoamónico	-	20 g
Sulfato de amonio	30 g	-
Sulfato de magnesio	114 g	30 g

Fuente: Elaborado por los Autores.

3.2. Etapa de Control

3.2.1. Control Fuzzy

El sistema de control Fuzzy utilizado es un sistema Mandami, fue diseñado utilizando lenguaje Python en una tarjeta Raspberry Pi 3, consiste en el sensor TDS, que mide la concentración de solutos en el agua circulante, valor que se convierte en un nivel de pertenencia para cada regla del sistema difuso. Luego en el controlador se realizan las operaciones difusas correspondientes y activa las bombas para conseguir la combinación de agua y solución nutritiva de acuerdo al nivel preestablecido.

Se distinguen entonces las variables lingüísticas de entrada y de salida. Las variables lingüísticas de entrada son valores medidos en el agua circulante. En huertos hidropónicos se manejan como variables de entrada el potencial de hidrogeniones (pH), la temperatura y la electroconductividad. Para el controlador difuso desarrollado en este proyecto se tomó la EC como variable a controlar.

Las variables de salida, son aquellas que van a manipular el sistema para alcanzar el valor preestablecido para la variable lingüística de entrada. En este sentido se identificó entre las variables de salida el agua salina y el agua neutra. Cuando el sensor de EC ubicado en el tanque de agua recirculante indica valores distintos a los establecidos, el controlador automáticamente enciende las bombas para agregar agua salina o neutra, según el requerimiento. La programación del controlador se realizó en lenguaje Python dentro de la tarjeta Raspberry Pi se encuentran en anexos. A continuación, se describen los valores asignados a las variables lingüísticas para la elaboración del universo de discurso.

Sensor	Rango	Unidades
Electroconductividad	0 - 1000	Partes por millón
Función de membresía	Tipo de Función	Valores
Muy disuelto	Trapecio	-1000, -1000, -750, -250
Poco disuelto	Triangular	-750, -250, 0.0
Neutro	Triangular	-400, 0.0, 400
Poco concentrado	Triangular	0.0, 250, 750
Muy concentrado	Trapecio	250, 750, 1000, 1000

Tabla 3: Funciones de membresía para la variable de entrada

Fuente: Elaborado por los Autores.

Las funciones de membresía para las variables de salida se ejecutan por la velocidad de giro de las bombas peristálticas que suministran agua potable y solución nutritiva disuelta. Quedaron de la siguiente forma:

Variable	Rango PWM	Unidades
Agua Potable	-255 - 0	-
Solución Nutritiva	0 - 255	-
Función de membresía	Tipo de Función	Valores
Reducir mucho EC	Trapecio	-255, -255, -220, -180
Reducir poco EC	Triangular	-180, -120, .40
Mantener EC	Triangular	-80, 0.0, 80
Elevar poco EC	Triangular	40, 120, 180
Elevar mucho EC	Trapecio	180, 220, 255, 255

Tabla 4: Funciones de membresía para la variable de salida

Fuente: Elaborado por los Autores.

El valor de error corresponde a la diferencia entre el valor seteado y el valor medido, mientras que el cambio de error es la diferencia entre el error actual y el último error medido. Las reglas que sigue el controlador diseñado para el proyecto se guían por la comparación entre el error y el cambio de error para activar el PWM de las bombas peristálticas de agua potable y solución nutritiva. La lógica del controlador fuzzy se describe en la tabla.

	Cambio de error		
Error	Negativo	cero	positivo
Negativo	Muy negativo	negativo	Negativo
Cero	Cero	cero	Cero
Positivo	Positivo	positivo	Muy positivo

Tabla 5: Lógica difusa del controlador diseñado

Fuente: Elaborado por los Autores.

Control Fuzzy en Python (Mandami)

```
def leerSensores():
    PH = 3  # Leer valor del sensor de PH
    Sal = 900  # Leer valor del sensor de salinidad
    return PH, Sal

# Función para setear el tiempo de apertura de las válvulas


def setValves(AP, Nu):
    # Setear el tiempo de apertura del agua potable
    # Setear el tiempo de apertura del nutriente
    print(AP, Nu)
```

Figura 19: Programación de control fuzzy desde la Raspberry

Fuente: Elaborado por los Autores.

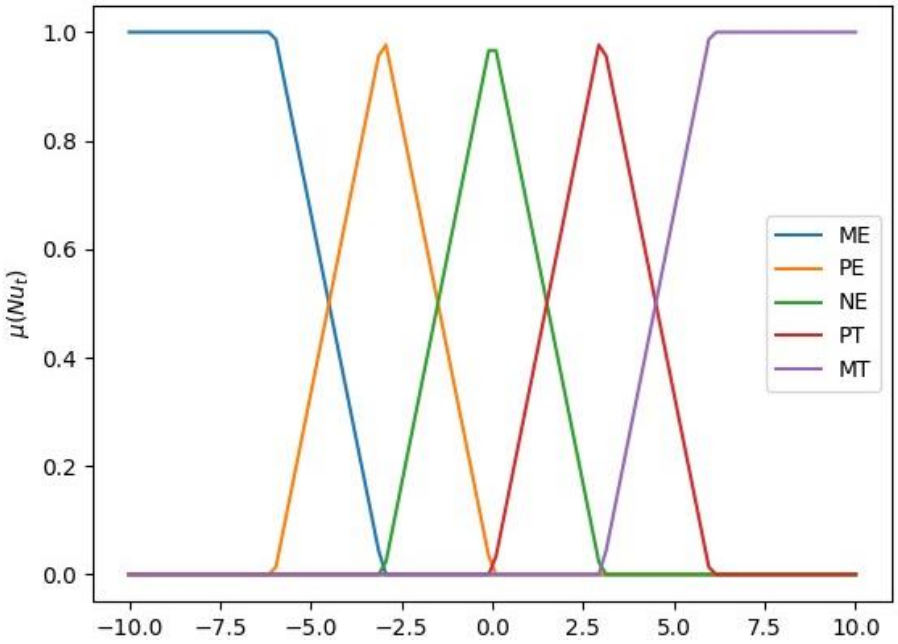


Figura 20: Tiempo de suministro de nutrientes

Fuente: Elaborado por los Autores.

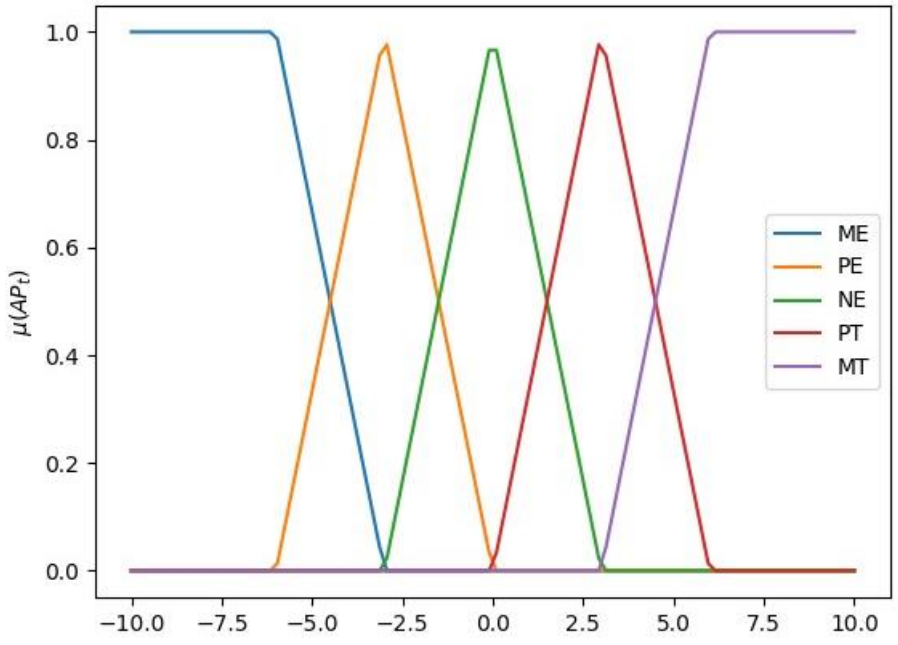


Figura 21: Tiempo de suministro de agua potable

Fuente: Elaborado por los Autores.

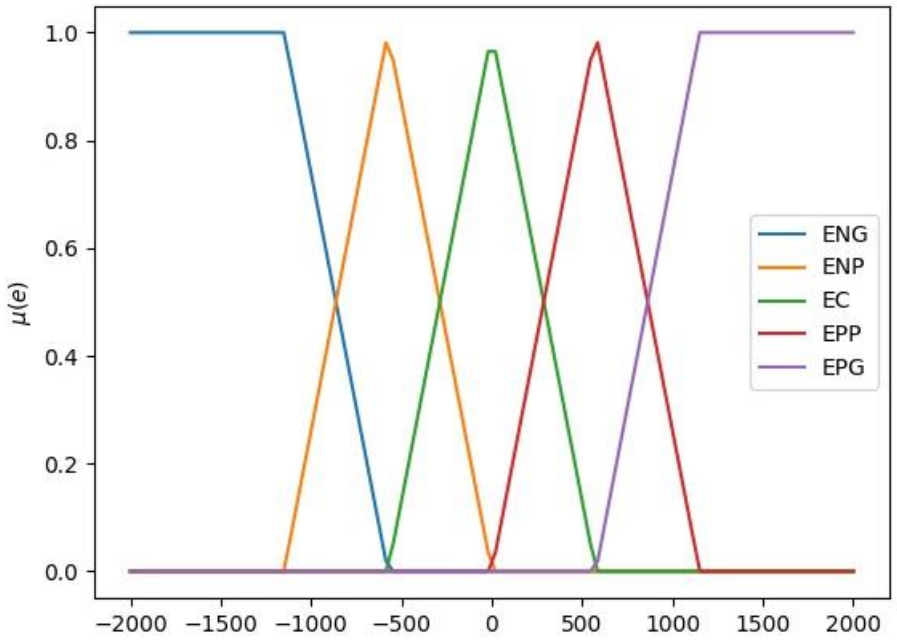


Figura 22: Error de salinidad del agua con nutrientes

Fuente: Elaborado por los Autores.

3.2.2. Adquisición de datos

El requerimiento de información del sistema implementado en este proyecto es básicamente el de los sensores instalados. Esta data sirve para el control difuso, pero además son indicadores para el usuario de que el sistema se encuentra en funcionamiento óptimo. En la tarjeta ESP32 de Arduino se programó en lenguaje Python utilizando el protocolo MQTT para que esta sirva de puente de comunicación entre el prototipo y una aplicación web diseñada con Node-Red para enviar y recibir datos desde la nube. Esto no sólo permitió revisar los parámetros en tiempo real desde cualquier dispositivo portátil o computadora con acceso a internet, sino que además permitió controlar el proceso, desde la modificación del valor establecido

hasta el manejo manual de cada bomba peristáltica de manera individual. El programa de Arduino ESP32 fue añadido a anexos. En la Figura 23 se observa cómo se accede a la interfaz gráfica de la raspberry desde una ventana de comandos.

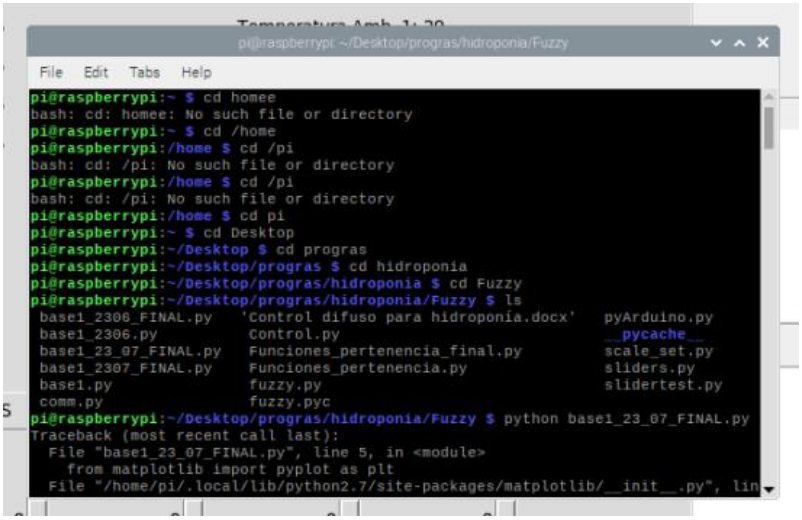


Figura 23: acceso a la interfaz gráfica desde la Raspberry

Fuente: Elaborado por los Autores.

Las interfaces gráficas diseñadas para el modelo de huerto hidropónico tienen soporte en el Sistema Operativo Raspbian al ejecutarse en la Raspberry Pi (Figura 24Figura 24), y en Node-Red para navegadores web (Figura 25). En ambas es posible encender y apagar los relés de las bombas peristálticas, así como dar marcha y paro a cada una de ellas de manera independiente en el modo manual. En el modo automático la programación permite configurar el valor de EC requerido en el sistema y el controlador fuzzy entra en funcionamiento para llevar la variable al valor requerido por el usuario. La interfaz web permite además observar gráficas

del funcionamiento del controlador Fuzzy y parámetros de funcionamiento de los motores. Su funcionamiento y características se describen en el siguiente apartado.

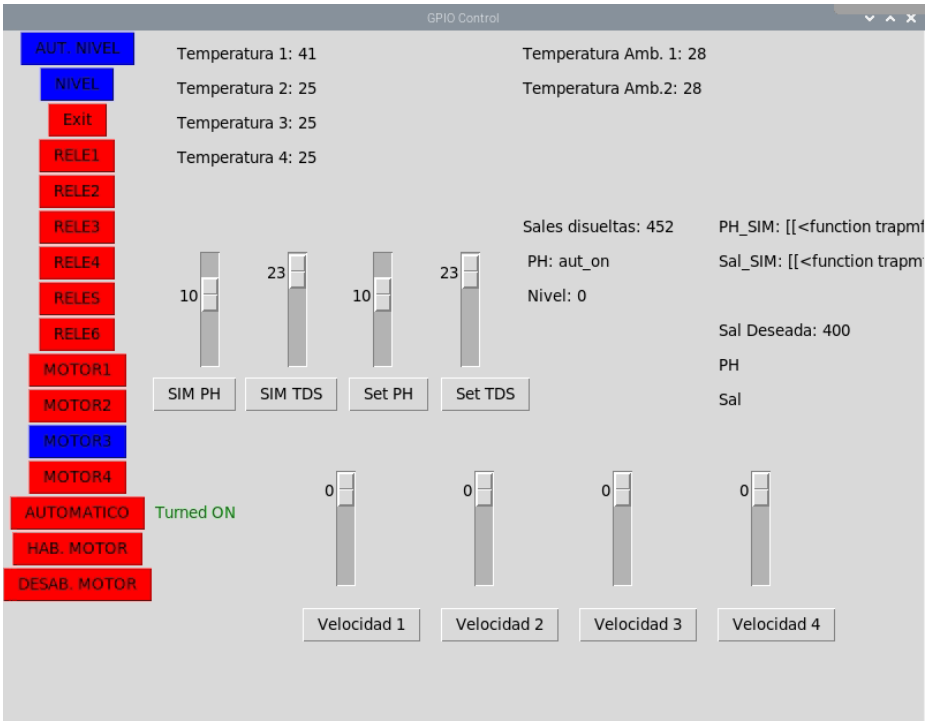


Figura 24: Interfaz gráfica en la Raspberry Pi

Fuente: Elaborado por los Autores.

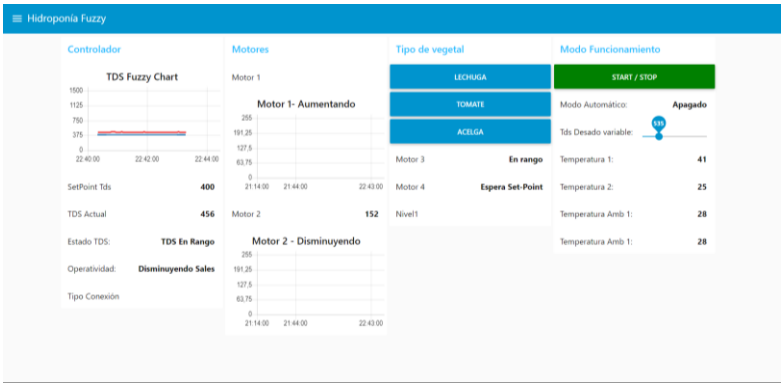


Figura 25: Interfaz gráfica de la aplicación web - Hidroponía Fuzzy

Fuente: Elaborado por los Autores.

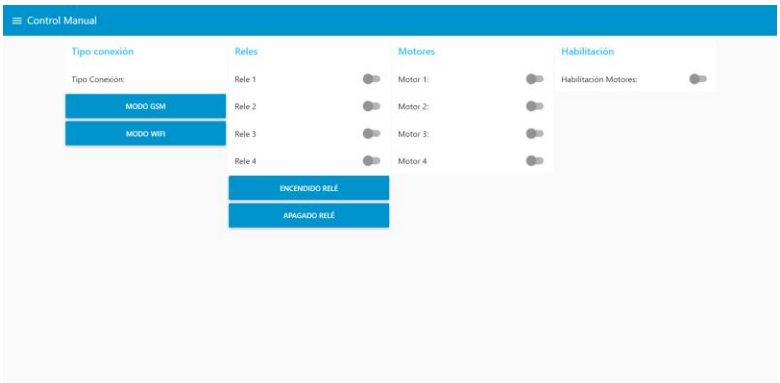


Figura 26: Interfaz gráfica de la aplicación web - Control Manual

Fuente: Elaborado por los Autores.

3.2.3. Interfaz Gráfica con Node-Red

La herramienta de diseño Node-Red instalada en la Raspberry permitió generar una interfaz gráfica amigable con el usuario que fue enviada a una dirección IP local para ser ejecutada en cualquier dispositivo conectado a la red. Mientras que la programación de diagrama de flujo se agregó a anexos, en la Figura 25 se muestra la interfaz gráfica como tal que se describe a continuación. El Dashboard o pantalla principal del aplicativo web creado como interfaz gráfica consta de dos ventanas. La principal llamada “Hidroponía Fuzzy” se observa en la Figura 27.

Figura 27: Interfaz Gráfica Web - Hidroponía Fuzzy

Fuente: Elaborado por los Autores.

3.2.4. Tarjeta de adquisición de datos

La PCB fue diseñada con el software KiCAD utilizando el esquema thru-hole, en el cual se utilizan componentes con pines metálicos. Estos pines atraviesan los terminales de la baquelita que previamente ya fue diseñada e impresa con las pistas que forman el circuito deseado. En la Figura 29 se observa el diseño esquemático de la PCB, que contiene borneras para conectar motores, sensor TDS y la ESP32.

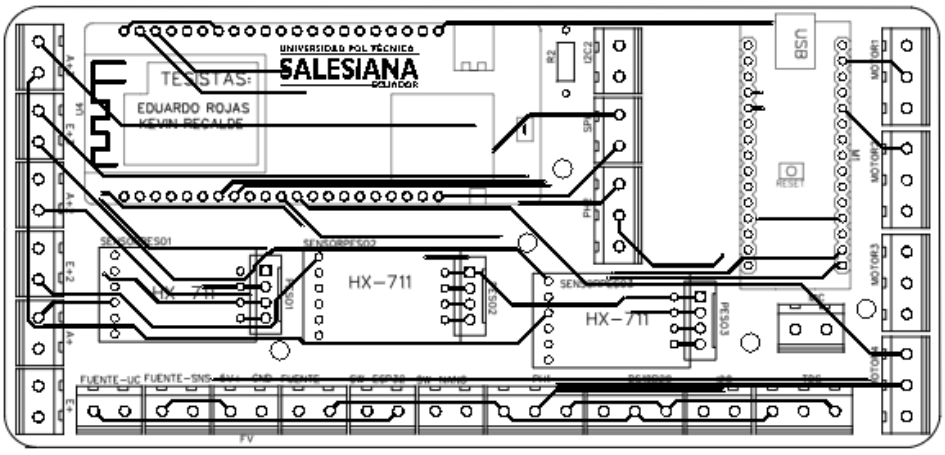


Figura 28: Vista PCB de tarjeta de adquisición de datos

Fuente: Elaborado por los Autores.

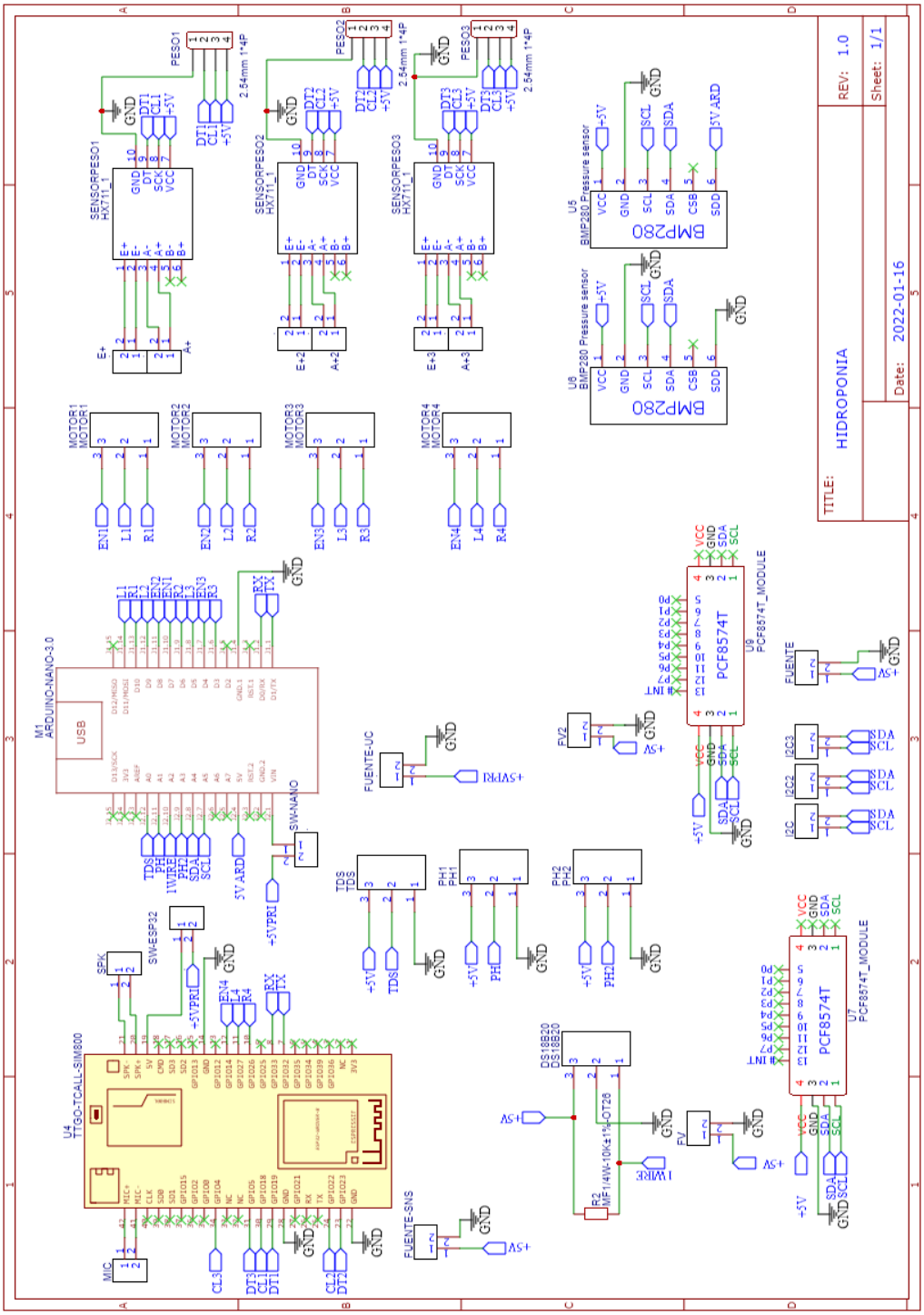


Figura 29: Diagrama esquemático de tarjeta de adquisición de datos

Fuente: Elaborado por los Autores.

3.2.1. Redundancia GSM

La ranura para tarjeta SIM integrada de la Arduino ESP32 brindó el beneficio de utilizar la red de datos móviles de Claro para la sincronización del módulo de hidroponia con la nube. Para brindar el servicio IoT de manera permanente se incluyó en la programación de la Arduino ESP32 líneas de código que activan el servicio GPRS cuando no hay señal de Wi-Fi disponible. Al reestablecerse el servicio de Wi-Fi es posible presionar un botón en la interfaz gráfica que reinicia el sistema para que deje de utilizar el servicio de red móvil y vuelva a conectarse al router.

CAPÍTULO IV

RESULTADOS Y ANÁLISIS

El módulo hidropónico cumple con el proceso de control tanto para un agua recirculante con concentración de sales baja, como con una de concentración de sales alta. Se realizaron las pruebas con un valor establecido de 400. El primer caso vimos un sistema con agua recirculante de baja salinidad, y se aprecia en la Figura 30 como el motor 1 que inocula solución nutritiva en el sistema trabaja en valores de PWM que fluctúan entre 145 y 240.

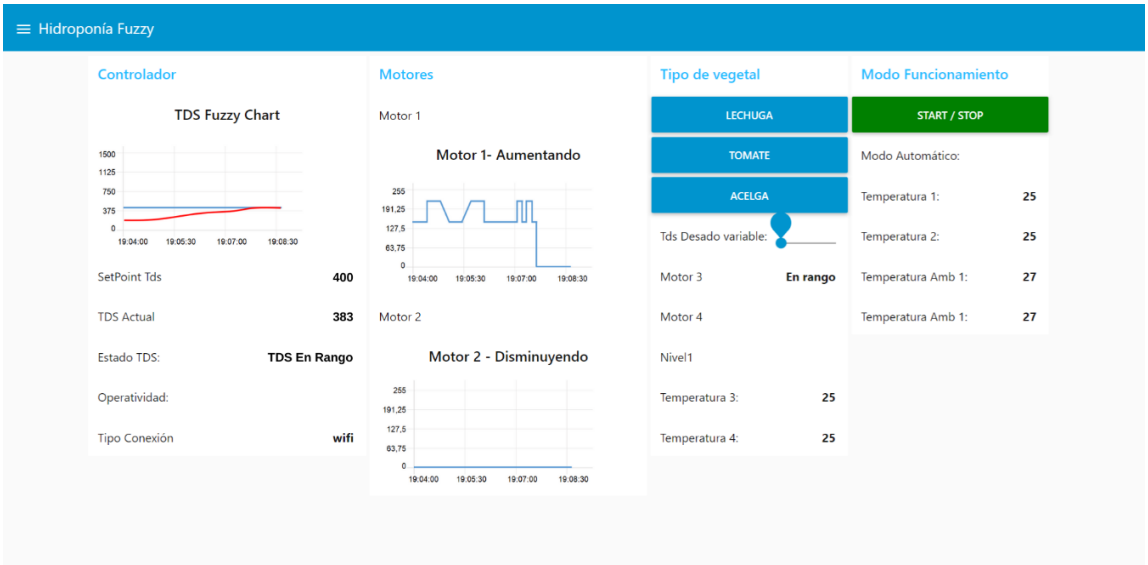


Figura 30: Control de Sistema con TDS bajo.

Fuente: Elaborado por los Autores.

En el caso de agua recirculante con TDS bajo, el sistema demoró para este intento un tiempo de 3 minutos con 45 segundos. En la siguiente experimentación se incluyó una perturbación que elevó el TDS del agua recirculante y el sistema actuó encendiendo el motor 2 como se observa en la Figura 31. Para este intento se observa que el controlador nivela el TDS en 11 minutos con 50 segundos.

Figura 31:Control de Sistema con TDS elevado

Fuente: Elaborado por los Autores.

Lo más notorio al momento de poner en funcionamiento el controlador difuso es que no actúa de manera veloz como ocurre en el caso de un controlador PID. Sin embargo, no existe requerimiento en el proceso que se está aplicando, dado que los minutos que toma nivelar los nutrientes en el agua no afectan de ninguna manera a las plantas en cualquiera de sus estados.

Al momento de programar el error con el que funcionan las bombas peristálticas se colocó en la variable TDS un margen de +/- 50 ppm. Esto ocasiona que cuando el sensor de TDS detecta que el flujo de agua recirculante se acerca dentro del margen mencionado deja de accionar el funcionamiento de las bombas. Esto se ve reflejado en que cuando el valor TDS Actual del Dashboard entra al margen el Estado TDS cambia a TDS En Rango y el controlador deshabilita las bombas.

4.1. Práctica #1: Manejo de condicionales con Node-Red

<table>
<tr><td colspan="3">FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN – PARA DOCENTES</td></tr>
<tr><td colspan="2">CARRERA:</td><td>ASIGNATURA:</td></tr>
<tr><td>NRO. PRÁCTICA:</td><td>1</td><td>TÍTULO PRÁCTICA: Manejo de condicionales con Node-RED</td></tr>
<tr><td colspan="3">OBJETIVO:
• OBJETIVO GENERAL.
Agregar sentencias para varias acciones en el módulo hidropónico IoT.
• OBJETIVOS ESPECÍFICOS:
- Declaración de variables y uso de librerías para control de motores.
- Implementar un diagrama de flujo que utilice una sentencia de control de bomba peristáltica.</td></tr>
<tr><td colspan="2">INSTRUCCIONES:</td><td>1. Descargar el archivo el cual contiene las instrucciones.
2. Responder la tarea a través del AVAC, adjuntando un archivo en Raptor (extensión .rar) para cada ejercicio. Los archivos deben ser almacenados en una carpeta con su NOMBRE_APELLIDO, y adjuntados como respuesta al taller.</td></tr>
<tr><td colspan="3">ACTIVIDADES POR DESARROLLAR</td></tr>
<tr><td colspan="3">1. Se declara el actuador en el módulo BTS con su conexión en los pines respectivos.
#define FwrdEnMotRight 14
#define FwrdSpeedMotRight 26</td></tr>
</table>

```
#define RvrsSpeedMotRight 27
```

1. Se declara el tipo de variable que hace la acción que activará el motor. La librería #include <ArduinoJson.h> permite hacer un tratamiento de los datos para poder enviar y recibir el formato JSON

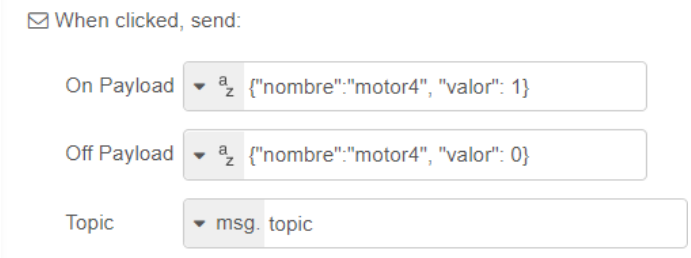
```
deserializeJson(docDisparadores, (const byte*)message, length);
```

2. Se separa en diferentes datos para que según su llegada. A partir de ahí una condicional verifica si es la secuencia de caracteres deseados para encender o apagar el motor.

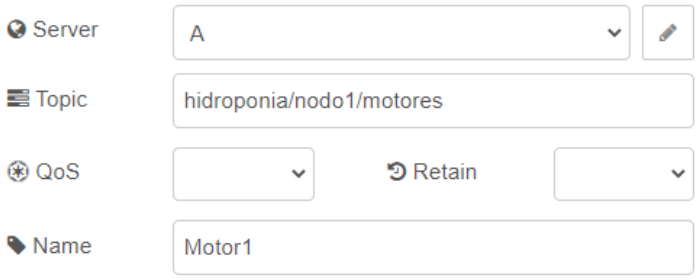
```
if(String(topic) == "hidroponia/nodo1/motores"){
if (docDisparadores["nombre"] == "motor4" && docDisparadores["valor"] == 1) {
int velMotor4 = 255;
digitalWrite(FwrdEnMotRight, HIGH);
ledcWrite(pwmChannel_0, velMotor4);          // FwrdSpeedMotRight
ledcWrite(pwmChannel_1, 0);          // FwrdSpeedMotLeft
   }
if (docDisparadores["nombre"] == "motor4" && docDisparadores["valor"] == 0) {
int  velMotor4 = 0;
digitalWrite(FwrdEnMotRight, LOW);
     //Serial.println("apagar  motor 1 ");
}
```

Permitiendo así su manejo, esos datos son enviados desde el control automáticamente según lo seteado en el control, o manualmente enviando la trama al topico deseado.

3. En node red se declaran los botones que al ser activados enviarán la trama

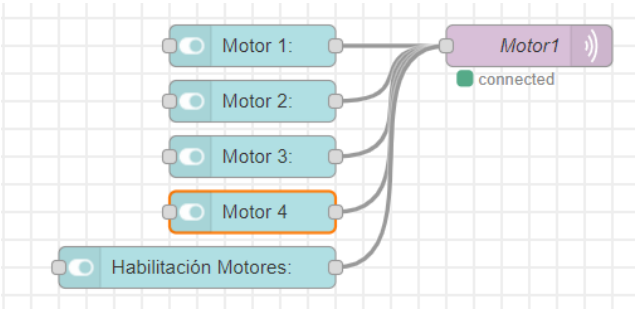


4. Luego se orienta al tema específico que fue declarado.



RESULTADO(S) OBTENIDO(S):

Se comprueba la puesta en marcha de los motores a través de los botones creados en el dashboard.



CONCLUSIONES:

Los estudiantes estarán en capacidad de agregar sentencias en Node-RED para la activación y desactivación de bombas peristálticas.

RECOMENDACIONES:

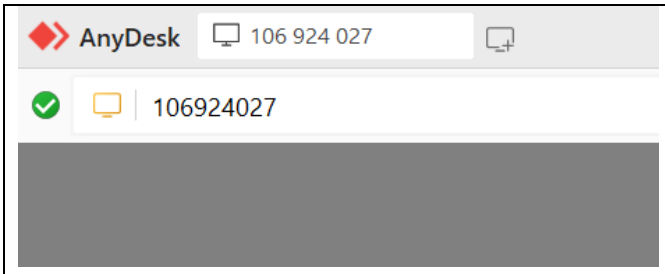
Verificar la correcta utilización de operadores, variables y expresiones, del lenguaje utilizado, para poder llevar a cabo la implementación.

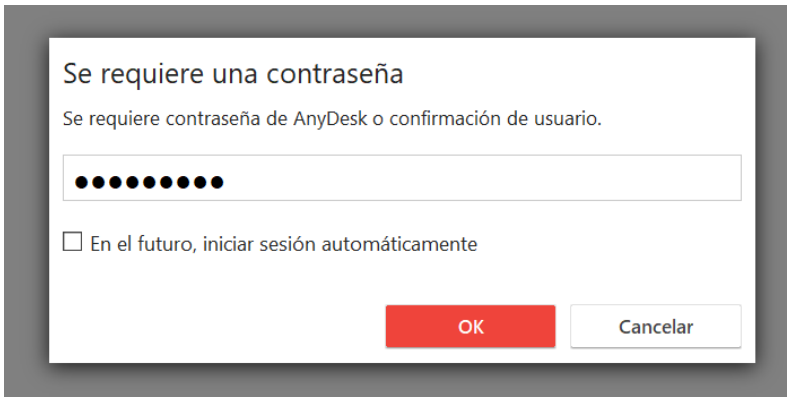
Docente: _____

Firma: _____

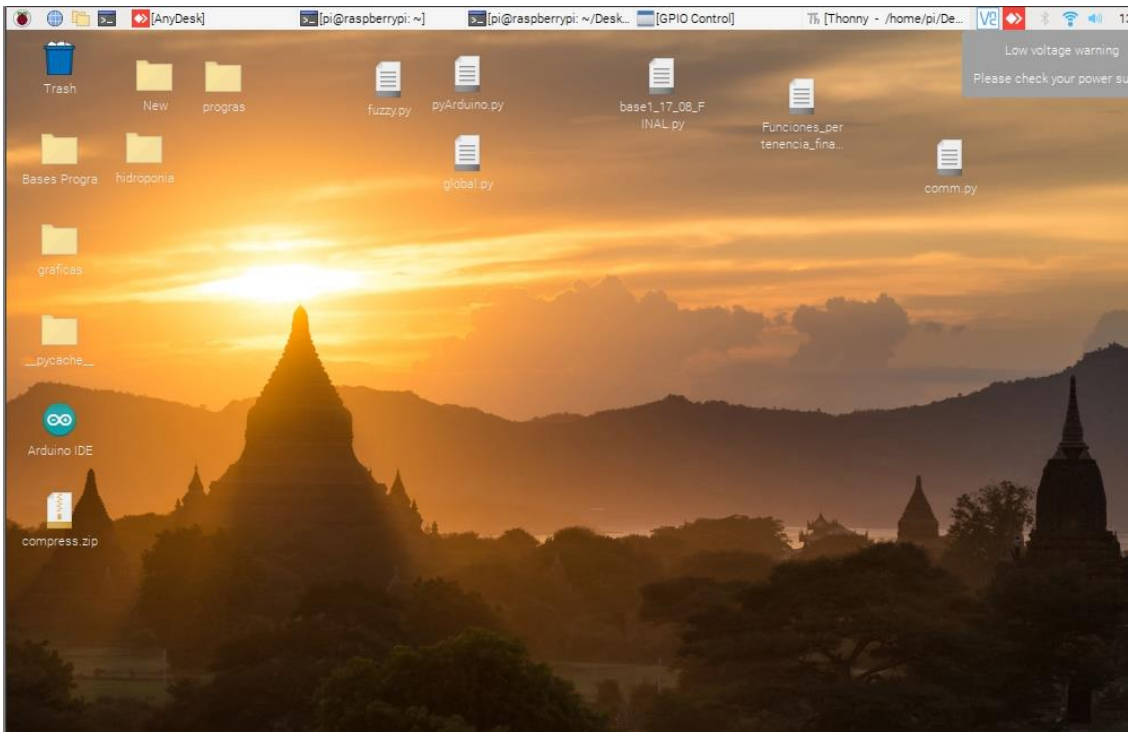
4.2. Práctica #2: Cambio de parámetros del controlador fuzzy

<table>
<tr><td colspan="4">FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN – PARA DOCENTES</td></tr>
<tr><td colspan="2">CARRERA:</td><td colspan="2">ASIGNATURA:</td></tr>
<tr><td>NRO. PRÁCTICA:</td><td>2</td><td colspan="2">TÍTULO PRÁCTICA: Cambio de Parámetros de Controlador Fuzzy</td></tr>
<tr><td colspan="4">OBJETIVO:
• OBJETIVO GENERAL.
Modificación de valores de las funciones de membresía.
• - OBJETIVOS ESPECÍFICOS:
- Acceder remotamente a la Raspberry del huerto hidropónico IoT.
- Modificar valores de los parámetros Fuzzy.</td></tr>
<tr><td colspan="2" rowspan="2">INSTRUCCIONES:</td><td colspan="2">5. Descargar el archivo el cual contiene las instrucciones.</td></tr>
<tr><td colspan="2">6. Responder la tarea a través del AVAC, adjuntando un archivo en Raptor (extensión .rar) para cada ejercicio. Los archivos deben ser almacenados en una carpeta con su NOMBRE_APELLIDO, y adjuntados como respuesta al taller.</td></tr>
<tr><td colspan="4">ACTIVIDADES POR DESARROLLAR</td></tr>
<tr><td colspan="4">1. Acceder a la Raspberry mediante el software Anydesk utilizando la dirección “106 924 027” y la contraseña por defecto “raspberry”.</td></tr>
</table>





2. Acceder al programa del sistema llamado “base1_17_08_FINAL.py” que se encuentra en el escritorio.



3. Las líneas 312 a 317 son las funciones de membresía de las variables de entrada. Para este caso la lectura del sensor de TDS.

```
311
312 Sal_ENG = [trapmf, [-1000, -1000, -750, -250]]
313 Sal_ENP = [trimf, [-250, -100, 0.0]]
314 #Sal_EC = [trimf, [-250, 0.0, 250]]
315 Sal_EC = [trimf, [-50, 0.0, 50]]
316 Sal_EPP = [trimf, [0.0, 100, 250]]
317 Sal_EPG = [trapmf, [250, 750, 1000, 1000]]
318
```

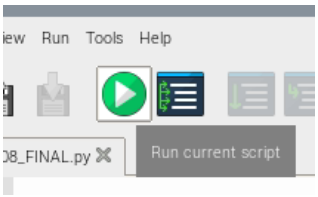
Se reconoce en el código primero el nombre de las funciones para las que se eligió la variable "sal" seguida de codificadores EN para error negativo, EC para error centro y EP para error Positivo. Con letras G y P al final para errores Grandes y Pequeños. Seguido de estas entre corchetes vemos comandos trapmf y trimf que llaman a las funciones de membresía de forma trapezoidal y triangular respectivamente. Los valores a continuación son los límites de izquierda a derecha de las funciones, según su forma tienen cuatro puntos la trapezoidal y tres puntos la triangular. Se procede a modificar estos valores por valores cercanos.

4. Luego en las líneas de código 347 a 351 observamos la declaración de las funciones de membresía para la variable nutrientes.

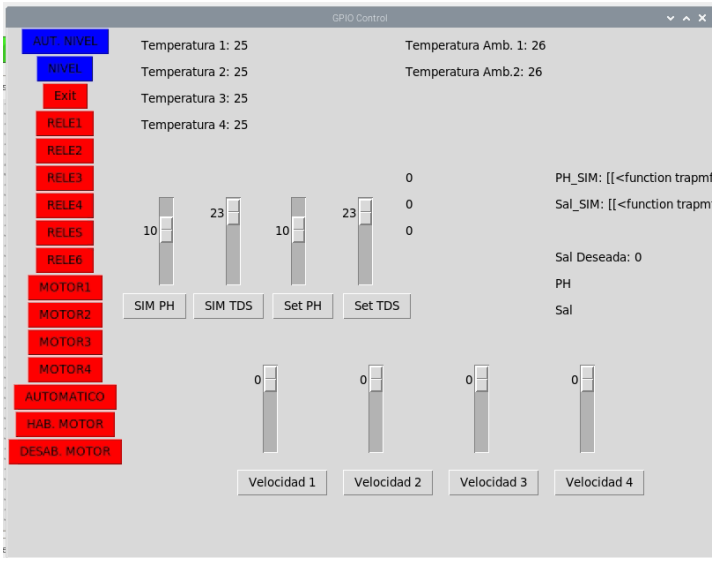
```
344
345 Nu_t = np.linspace(-255, 255, 100)
346 #255/7=36=40  80
347 Nu_ME = [trapmf, [-255, -255, -220, -180]]
348 Nu_PE = [trimf, [-180, -160, -120]]
349 Nu_NE = [trimf, [-120, 0, 120]]
350 Nu_PT = [trimf, [120, 160, 180]]
351 Nu_MT = [trapmf, [180, 220, 255, 255]]
352
353
```

La segmentación escogida para esta variable va en pasos de 100 y se maneja entre los valores de -255 a 255. Las funciones de membresía se declaran de igual manera que en el punto anterior. Procedemos a modificar estos valores por valores cercanos.

5. Compilamos el programa.



6. Encendemos el modo automático y recuperamos resultados de las gráficas en el dashboard de Node-RED.



RESULTADO(S) OBTENIDO(S):

Los estudiantes podrán comprobar el tiempo en el que el sistema llega al valor establecido deseado.

CONCLUSIONES:

Los estudiantes estarán en capacidad de modificar los parámetros del controlador en Python para creación de nuevos sistemas y controladores difusos.
RECOMENDACIONES: Los controladores difusos se diseñan con amplio conocimiento de los sistemas que desean ser controlados, por lo que requieren una amplia investigación previa de su funcionamiento.

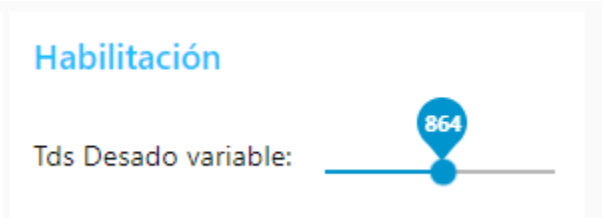
Docente: _____

Firma: _____

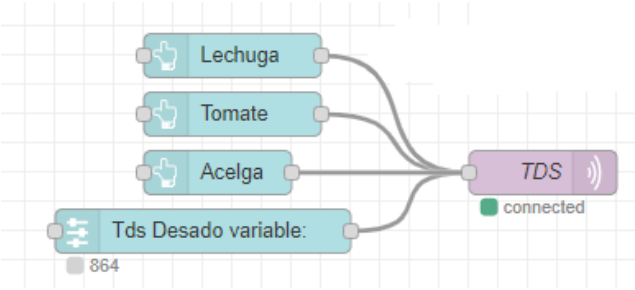
4.3. Práctica #3: Modificación de recetas

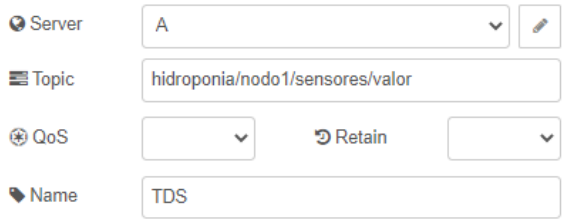
<table>
<tr><td colspan="4">FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN – PARA DOCENTES</td></tr>
<tr><td colspan="2">CARRERA:</td><td colspan="2">ASIGNATURA:</td></tr>
<tr><td>NRO. PRÁCTICA:</td><td>3</td><td colspan="2">TÍTULO PRÁCTICA: Modificación de recetas</td></tr>
<tr><td colspan="4">OBJETIVO:
• OBJETIVO GENERAL.
Agregar recetas con distintos parámetros de valor establecido.
• - OBJETIVOS ESPECÍFICOS:
- Agregar sentencias en diagrama de flujo que almacenen programas para sembrar otro tipo de vegetales.
- Crear botones en el dashboard de Node-RED.</td></tr>
<tr><td rowspan="2">INSTRUCCIONES:</td><td colspan="3">1. Descargar el archivo el cual contiene las instrucciones.</td></tr>
<tr><td colspan="3">2. Responder la tarea a través del AVAC, adjuntando un archivo en Raptor (extensión .rar) para cada ejercicio. Los archivos deben ser almacenados en una carpeta con su NOMBRE_APELLIDO, y adjuntados como respuesta al taller.</td></tr>
<tr><td colspan="4">ACTIVIDADES POR DESARROLLAR</td></tr>
</table>

1. Para declarar un valor establecido se hace click en alguna de las opciones o también se escoge específicamente la cantidad de sales disueltas deseadas.



2. En la pantalla del dashboard desarrollada en Node-RED se visualizan los programas unidos al sensor de TDS, tanto los preestablecidos como el variable.





3. Este dato será receptado por el sistema de control programado en Python, que toma los valores dentro del sistema Fuzzy.

```
if message.topic=="hidroponia/nodo1/sensores/valor":
     print("Mensaje recibido=", str(message.payload.decode("utf-8")))
     print("Llego TDS deseado")
     Sal_deseada = json.loads(mensaje)
     client.publish("hidroponia/nodo1/set_tds",Sal_deseada)#publish
     Sal_des_label.config(text="Sal Deseada: " +str(Sal_deseada))
```

RESULTADO(S) OBTENIDO(S):

CONCLUSIONES:

Los estudiantes estarán en capacidad de modificar los parámetros del controlador en Python para creación de nuevos sistemas y controladores difusos.

RECOMENDACIONES:

Los controladores difusos se diseñan con amplio conocimiento de los sistemas que desean ser controlados, por lo que requieren una amplia investigación previa de su funcionamiento.

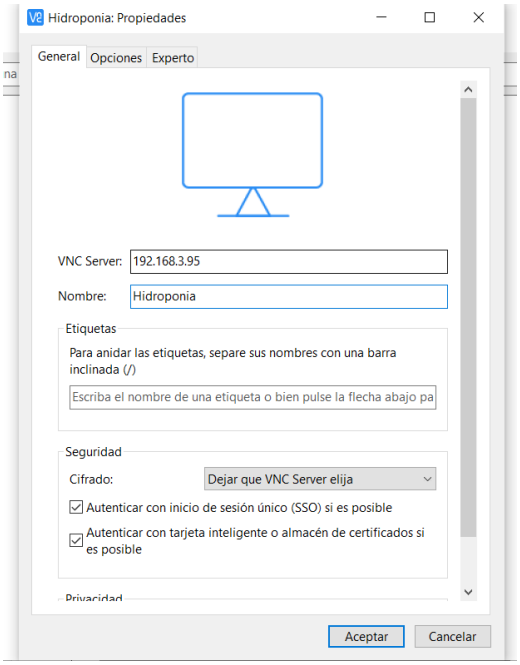
Docente: _____

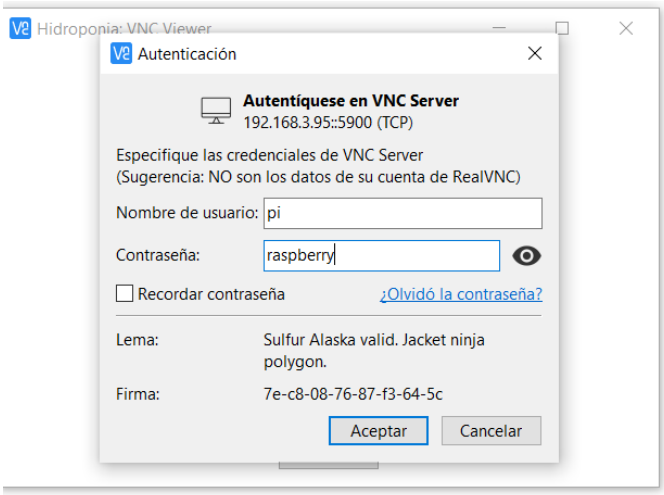
Firma: _____

4.4. Práctica #4: Tareas generales de la tarjeta electrónica

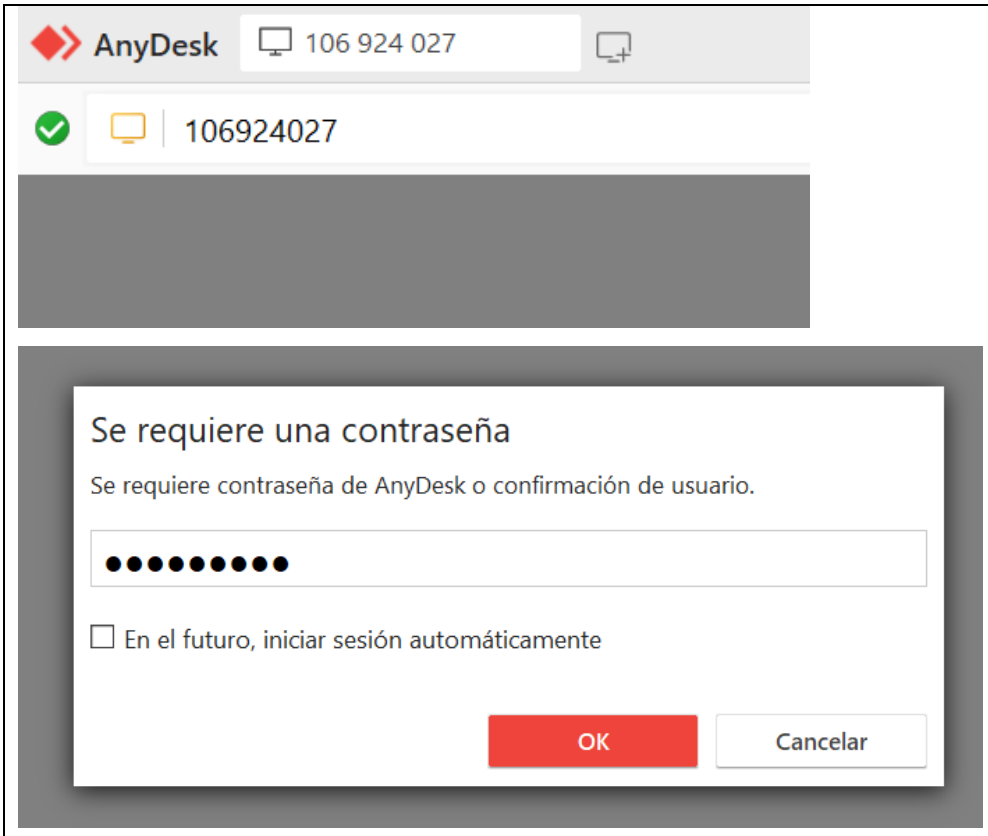
<table>
<tr><td colspan="3">FORMATO DE GUÍA DE PRÁCTICA DE LABORATORIO / TALLERES / CENTROS DE SIMULACIÓN – PARA DOCENTES</td></tr>
<tr><td colspan="2">CARRERA:</td><td>ASIGNATURA:</td></tr>
<tr><td>NRO. PRÁCTICA:</td><td>4</td><td>TÍTULO PRÁCTICA: Tareas Generales de la Tarjeta Electrónica</td></tr>
<tr><td colspan="3">OBJETIVO:
• OBJETIVO GENERAL.
Poner en marcha el módulo hidropónico semiautomático.
• - OBJETIVOS ESPECÍFICOS:
- Probar el funcionamiento manual del módulo.
- Controlar la salinidad del agua recirculante.</td></tr>
<tr><td colspan="2" rowspan="2">INSTRUCCIONES:</td><td>4. Descargar el archivo el cual contiene las instrucciones.</td></tr>
<tr><td>5. Responder la tarea a través del AVAC, adjuntando un archivo en Raptor (extensión .rar) para cada ejercicio. Los archivos deben ser almacenados en una carpeta con su NOMBRE_APELLIDO, y adjuntados como respuesta al taller.</td></tr>
<tr><td colspan="3">ACTIVIDADES POR DESARROLLAR</td></tr>
<tr><td colspan="3">1. Una vez conectado el módulo a una toma de 110 V AC se accede a la Rasberry por cualquiera de los siguientes métodos.</td></tr>
</table>

Utilizando el software VNC Viewer, utilizando la IP asignada por DHCP, colocando usuario y contraseña por defecto de la Raspberry.



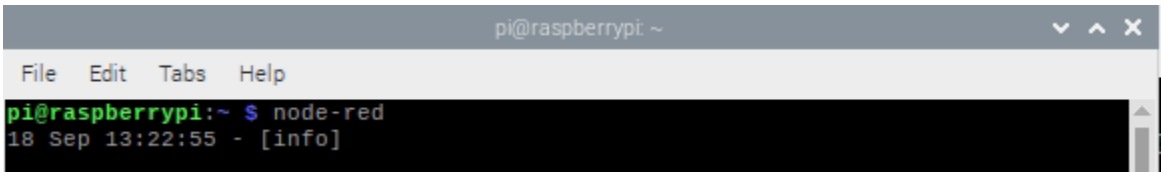


Utilizando el software Anydesk con la credencial “106924027” y la contraseña por defecto de la Raspberry.



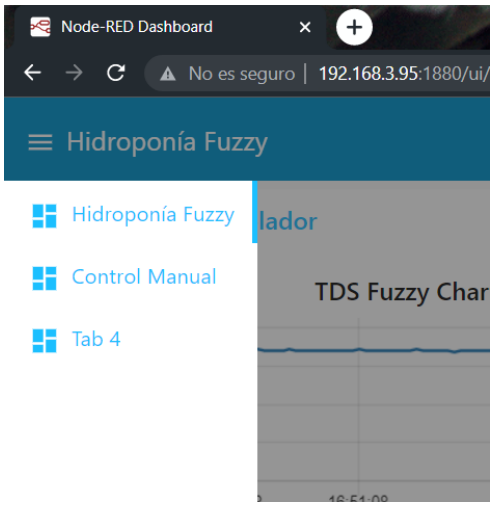
2. Abrir una ventana de comandos en la Raspberry e iniciar Node-RED utilizando la siguiente línea de código.

Node-red



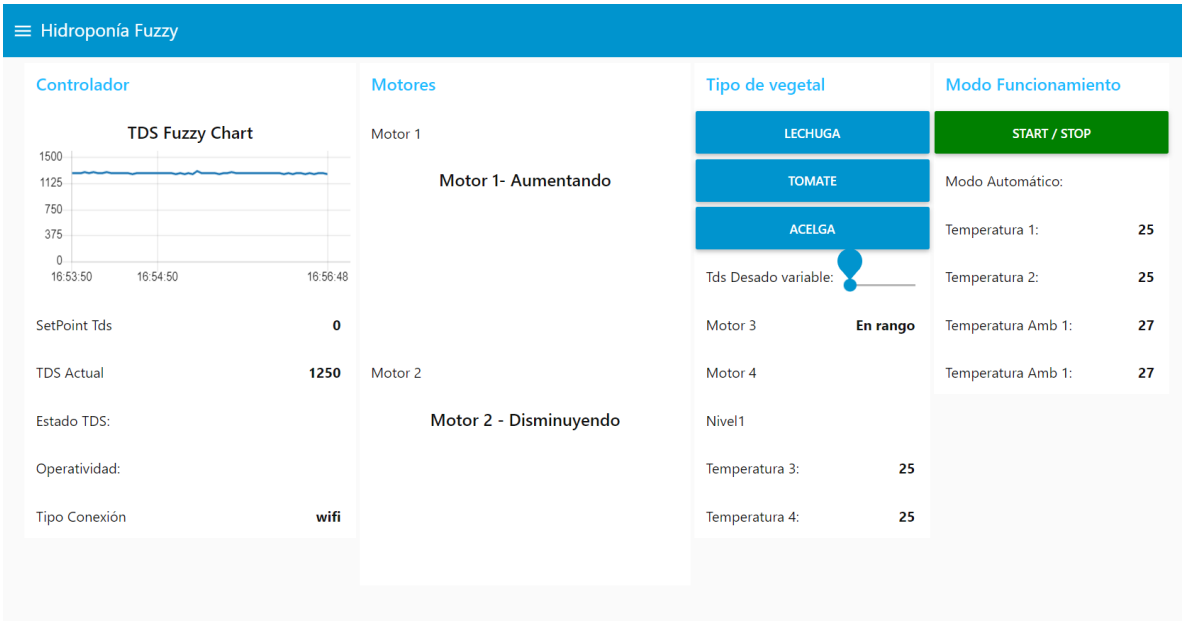
Esto inicia Node-RED, ahora podemos colocar la IP en un navegador que esté conectado a la red para acceder al Dashboard, ingresando la dirección 192.168.3.95:1880

3. Para acceder al control manual del módulo hidropónico damos click en el menú de la página y luego en control manual.



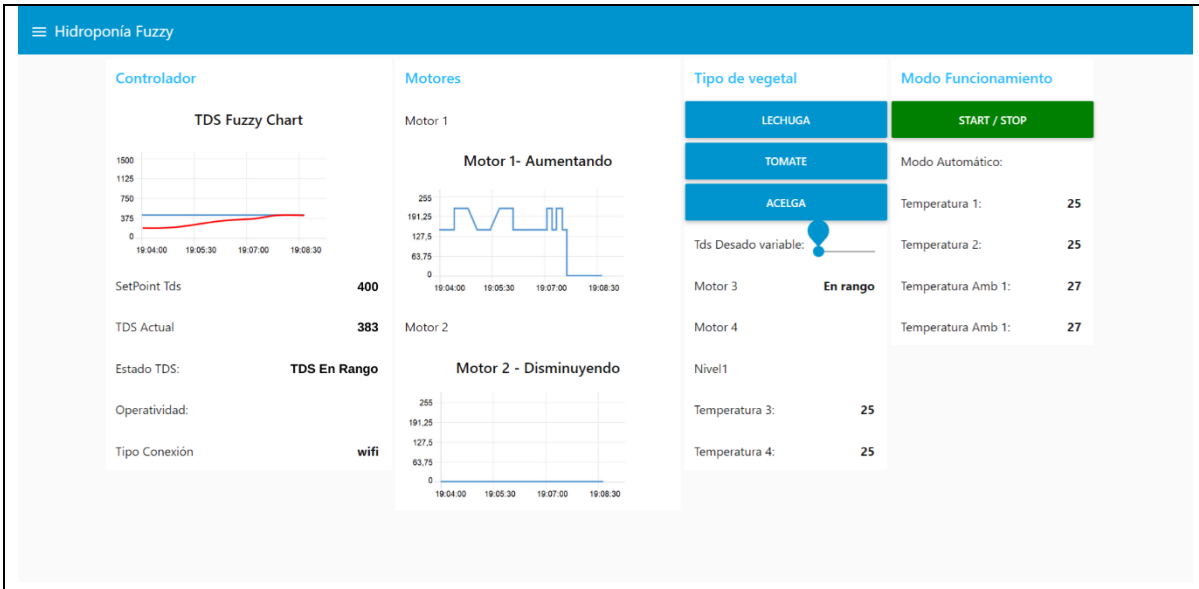
4. Dentro de control manual podemos encender los relés y habilitar las respectivas bombas peristálticas con los botones ya creados. De igual forma es posible escoger el tipo de conexión utilizada ya sea GSM o WIFI.

5. En la página principal del Dashboard podemos encender el modo automático, con el botón START/STOP, además de elegir el valor establecido o la receta de funcionamiento. También tenemos gráficos del controlador que muestra el TDS deseado y el actual, así como los motores que se encienden para corregir el error.



RESULTADO(S) OBTENIDO(S):

Se observó el funcionamiento del módulo para nivelar un sistema con TDS bajo y alto.



CONCLUSIONES:

Los estudiantes estarán en capacidad de utilizar el módulo, regular un nivel de TDS deseado y observar el funcionamiento del controlador.

RECOMENDACIONES:

El nivel de salinidad en el sistema alcanza el TDS deseado más fácilmente en subida que en bajada.

Al trabajar en un sistema donde fluye agua debe procurar no mojar los artefactos electrónicos con los que trabaje como celulares o laptops, así como las tarjetas electrónicas del módulo.

Docente: _____

Firma: _____

CONCLUSIONES

De acuerdo con el trabajo presentado, a los resultados obtenidos y su discusión, se pueden plantear las siguientes conclusiones:

- El diseño de sistema hidropónico NFT modular, facilita el acceso a cultivos con una inversión bastante inferior al costo de adquirir y mantener una propiedad para siembra convencional.
- El sistema IoT del módulo hidropónico fue diseñado con interfaces amigables con el usuario, lo que vuelve accesible el control de variables en tiempo real, permitiendo percibir y corregir cambios repentinos de parámetros al instante.
- Un control difuso de dosificación de nutrientes aporta la cantidad justa de nutrientes de acuerdo a la demanda del cultivo, además de facilitar el control para el usuario permitiéndole observar en tiempo real el estado del cultivo en función de los nutrientes que se dan y los que se necesitarían.
- El sistema de notificaciones redundantes por tecnología Wi-Fi y GSM permite una comunicación constante; la que fue puesta a prueba simulando una caída en la red de hogar mediante la desconexión del módem. Al no detectar Wi-Fi el módulo automáticamente se conecta al internet de la red celular.

  Las prácticas fueron diseñadas de manera didáctica y pedagógica, para describir a detalle el funcionamiento del módulo tanto a docentes como a estudiantes de la Universidad Politécnica Salesiana.

RECOMENDACIONES

- El proyecto se realizó con la confianza de encontrarse en un sitio con señal de internet celular, sin embargo, en lugares remotos o entornos más agresivos, es necesario orientar la planificación a soluciones de conexión robustas que se encuentren disponibles, siempre con el debido análisis económico y proyecciones de mantenimiento para la evaluación costo beneficio.
- Al programar es importante llevar un control de versiones que permita mantener orden en las diferentes etapas del desarrollo de proyecto, de igual manera en caso de que exista un error crítico tener un punto de partida desde el cual poder retomar sin perder la totalidad del esfuerzo dedicado.
- La búsqueda de proveedores confiables y equipos de buen estándar y calidad incrementa la escalabilidad del proyecto, puesto que evita que encontremos obstáculos en caso que se requiera reemplazar un elemento, ya sea por avería o para mejoras en el sistema.
- Luego de pasar el periodo de pruebas del prototipo, es recomendable buscar nuevos componentes de menor tamaño y con alta rotación de stock para obtener un resultado más práctico y económico, lo cual facilitaría su aplicación en otras investigaciones.

BIBLIOGRAFÍA

Andreau, R. (2015). Capítulo 5: Soluciones Nutritivas I. En J. Beltrano, *Cultivo en Hidroponia .* La Plata: Universidad Nacional de La Plata.

AVIA. (2020). 24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales. *HX711 datasheet*. Xiamen, P. R., China. Obtenido de https://cdn.sparkfun.com/datasheets/Sensors/ForceFlex/hx711_english.pdf

Beltrano, J. (2015). Capítulo I: Introducción al Cultivo en Hidroponía. En *Cultivo en hidroponía.* La Plata: Universidad Nacional de La Plata (UNLP).

Bosch. (2015). *BMP280 Digital Pressure Sensor Datasheet.* Obtenido de adafruit: https://cdn-shop.adafruit.com/datasheets/BST-BMP280-DS001-11.pdf

Carbone, A. (2015). Capítulo 4: Nutrición Mineral . En J. Beltrano, *Cultivo en Hidroponia.* La Plata: Universidad Nacional de La Plata.

Castro, L. (22 de febrero de 2016). *Lógica Difusa*. (Universidad Virtual de América) Obtenido de https://univia.info/la-logica-difusa/

Giménez, D. (2015). Capítulo III: Economía del carbono. En *Cultivo en hidroponía.* La Plata: Universidad Nacional de La Plata (UNLP) .

GroHo. (2021). *La Importancia del CO2 en la Hidroponia*. (GroHo) Obtenido de https://www.groho.es/post/importancia-del-co2-en-hidroponia

Hurtado, J. (2014). *Lógica difusa: perspectiva y aplicaciones.* Quindío: Universidad del Quindío. Obtenido de https://core.ac.uk/reader/287137100

Llamas, L. (2022). *Zona Geek*. Obtenido de ¿QUÉ ES MQTT? SU IMPORTANCIA COMO PROTOCOLO IOT: https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/

Llano, L. (2007). Sistema de Inferencia Difuso para Identificar Eventos de Falla en Tiempo Real del STE usando Registros SOE. *Avances en Sistemas e INformática, 4*(2). Obtenido de https://www.researchgate.net/publication/220136818_Sistema_de_Inferencia_Difuso_para_Identificar_Eventos_de_Falla_en_Tiempo_Real_del_STE_usando_Registros_SOE

Mashhad, A. (2013). PID like fuzzy logic control of an Unmanned Underwater Vehicle. *13th Iranian Conference on Fuzzy Systems, IFSC*, (págs. 1-5). Obtenido de https://www.researchgate.net/publication/261489402_PID_like_fuzzy_logic_control_of_an_Unmanned_Underwater_Vehicle

Maxim Integrated. (2019). Programmable Resolution 1-Wire Digital Thermometer. San Jose, California, USA. Obtenido de DS18B20: https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

Mosquera, P. (2014). *Operación confiable de una microrred aislada con generador principal diesel, aplicando un sistema de control difuso en el deslastre de sus cargas.* Universidad Politécnica Salesiana. doi:https://dspace.ups.edu.ec/handle/123456789/7290

MovilTronics. (2021). *Moviltronics*. Obtenido de Modulo driver puente h para motor de alta potencia: https://moviltronics.com/tienda/modulo-driver-puente-h-para-motor-de-alta-potencia-ibt2-43a/

Naylamp. (2019). *Naylamp mechatronics*. Obtenido de SENSOR DE TEMPERATURA DIGITAL DS18B20: https://naylampmechatronics.com/sensores-temperatura-y-humedad/16-sensor-de-temperatura-digital-ds18b20.html

Node-RED. (2022). *Node-RED* . Obtenido de Node-RED ORG: https://nodered.org/

Noguès, O. (2021). *merveilles-du-monde.com: "Les jardins suspendus de Babylone"*. Obtenido de https://www.merveilles-du-monde.com/Sept/Description-des-jardins.php

Ogata, K. (2010). *INGENIERIA DE CONTROL MODERNA.* Madrid: Ed. Pearson Educación S.A., 5 ed.

Raspberry Pi. (2021). *Raspberry Pi 4 modelo B*. Obtenido de Raspbery pi org: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/

Ravi, L., & Mohamed, D. (2020). Automated smart hydroponics system using internet of things. *International Journal of Electrical and Computer Engineering (IJECE), 10*(6389-6398). doi:http://ijece.iaescore.com/index.php/IJECE/article/view/20870/14427

Robotics ORG. (2021). *ESP32 GSM*. Obtenido de Robotics org: https://www.robotics.org.za/TTGO-ESP32-GSM

Rogel, M. (2018). *Respuesta de tres variedades de lechuga a cuatro soluciones nutritivas, bajo condiciones hidropónicas en invernadero.* Quito: Universidad Central del Ecuador. Obtenido de http://www.dspace.uce.edu.ec/bitstream/25000/16529/1/T-UCE-0004-CAG-024.pdf

Ruscitti, M. (2015). Capítulo II: El agua en la planta. En *Cultivo en hidroponía.* La Plata: Universidad Nacional de La Plata (UNLP).

Sanitron. (2019). *Válvulas Solenoide*. (Sanitron) Obtenido de https://sanitronec.com/producto/valvulas-solenoide/

Tendenzias. (20 de 05 de 2019). *sobrehistoria.com - "Los Jardines Colgantes de Babilonia"*. Obtenido de https://sobrehistoria.com/los-jardines-colgantes-de-babilonia/

Unitronic. (2020). *Unitronic*. Obtenido de MODULO CONTROLADOR XFW-HX711 SENSOR PESO: https://unitronic-online.com/producto/modulo-controlador-xfw-hx711-sensor-peso/

Urdiales, V., & Espín, J. (2018). Monitoreo de un sistema hidropónico NFT a escala usando arquitectura arduino. *Tecnología en Marcha, 31*(2), 147-158. Obtenido de https://www.scielo.sa.cr/pdf/tem/v31n2/0379-3982-tem-31-02-147.pdf

Watson Marlow. (2022). *Watson Marlow - Fluid Technology Solutions*. Obtenido de ¿Cómo funcionan las bombas peristálticas? Sanitarias:

https://www.wmftg.com/es-es/support/how-do-peristaltic-pumps-work-sanitary/

Yang, S.-H. (2013). Internet of Things. En S.-H. Yang, *Wireless Sensor Networks* (págs. 247-261). London: Springer. doi:https://link.springer.com/chapter/10.1007/978-1-4471-5505-8_12

ANEXOS

5.1. Programación Arduino ESP32

```
#define TINY_GSM_MODEM_SIM800
#define SerialMon Serial
#ifndef __AVR_ATmega328P__
#define SerialAT Serial1

// or Software Serial on Uno, Nano
#else
#include <SoftwareSerial.h>
SoftwareSerial SerialAT(2, 3);  // RX, TX
#endif
#include <Preferences.h>
Preferences preferences;
// See all AT commands, if wanted
// #define DUMP_AT_COMMANDS

// Define the serial console for debug prints, if needed
#define TINY_GSM_DEBUG SerialMon

// Range to attempt to autobaud
// NOTE:  DO NOT AUTOBAUD in production code.  Once you've established
// communication, set a fixed baud rate using modem.setBaud(#).
#define GSM_AUTOBAUD_MIN 9600
#define GSM_AUTOBAUD_MAX 115200

// set GSM PIN, if any
#define GSM_PIN ""

// Your GPRS credentials, if any
const char apn[]      = "internet.claro.com.ec";
```

```
const char gprsUser[] = "";
const char gprsPass[] = "";

// SIM card PIN (leave empty, if not defined)
const char simPIN[]   = "";

// TTGO T-Call pins
#define MODEM_RST            5
#define MODEM_PWKEY          4
#define MODEM_POWER_ON       23
#define MODEM_TX             27
#define MODEM_RX             26
#define I2C_SDA              21
#define I2C_SCL              22
```

```
#include <ArduinoJson.h>//libreria para serializar valores de Presion
//Librerias para celdas decarga
#include <TinyGsmClient.h>
#include <HX711_ADC.h>

//Librerias para WIfi y Mqtt
#include <WiFi.h>
#include <PubSubClient.h>
#include <WiFiClient.h>
#include <Wire.h>
```

```
#ifdef DUMP_AT_COMMANDS
#include <StreamDebugger.h>
StreamDebugger debugger(SerialAT, SerialMon);
TinyGsm        modem(debugger);
#else
TinyGsm        modem(SerialAT);
#endif
#include <Adafruit_Sensor.h>
// I2C for SIM800 (to keep it running when powered from battery)


TinyGsmClient client(modem);
//PubSubClient  mqtt(client);
PubSubClient  mqtt(client);
WiFiClient espClient;
long lastMsg = 0;
//PubSubClient mqttwifi(espClient);
int cont = 0;
uint32_t lastReconnectAttempt = 0;
uint32_t lastReconnectAttempt2 = 0;

String tipo;
#define uS_TO_S_FACTOR 1000000UL   /* Conversion factor for micro seconds to seconds */
#define TIME_TO_SLEEP  3600        /* Time ESP32 will go to sleep (in seconds) 3600 seconds = 1 hour */

#define IP5306_ADDR          0x75
#define IP5306_REG_SYS_CTL0  0x00
```

```
//Variables para declarar segunda COM serial
#define RxArdu 32
#define TxArdu 33
//#define disparadorNano 32

//33 25 26 27 14 12
//Pines de balanzas
#define pinDatPeso1 34
#define pinCkPeso1  35
#define pinDatPeso2 25
#define pinCkPeso2  26
#define pinDatPeso3 27
#define pinCkPeso3  14

int valTemp1, valTemp2, valTemp3, valTemp4;
int valPeso1, valPeso2, valPeso3, valPeso4;
int valTempAmb1, valTempAmb2, valPresAmb1, valPresAmb2;
int valTdsAgua;
int valPh;


//Variables para conexion Wifi y Mqtt
const char* ssid = "NOMBREDELAREDWIFI";
const char* password = "CONTRASEÑADELAREDWIFI";

const char* broker = "178.62.53.149";
const char* idClient = "nodo1prueba";
```

```
const char* userMqtt = "web_client";
const char* passMqtt = "121212";
int portMqtt = 1883;
String modoConecnt = "";

//idClient, userMqtt, passMqtt



//Documentos para los Objetos Json
DynamicJsonDocument docSensoresRx(256);
DynamicJsonDocument docTemp(512);
DynamicJsonDocument docPeso(512);
DynamicJsonDocument docBMP(512);
DynamicJsonDocument docTDS(48);
DynamicJsonDocument docPH(48);
//Documento Json para los mensajes recibidos
DynamicJsonDocument docDisparadores(512);

//docTemp, docPeso, docBMP, docTDS, docPH



//Objetos de celdas decarga

HX711_ADC LoadCell_1(pinDatPeso1, pinCkPeso1); //HX711 1
HX711_ADC LoadCell_2(pinDatPeso2, pinCkPeso2); //HX711 1
HX711_ADC LoadCell_3(pinDatPeso3, pinCkPeso3); //HX711 1



/*
hidroponia/nodo1/disparadores/#
```

```
hidroponia/nodo1/disparadores/modulosexpan/modulo1
hidroponia/nodo1/disparadores/modulosexpan/modulo2
hidroponia/nodo1/disparadores/motores
*/

// Pines y configuraciones para control de motor4 por PWM
const int freq              = 500;
const int pwmChannel_0      = 0;
const int pwmChannel_1      = 1;
const int resolution        = 8; //(8 bits -> 0-255)

#define FwrdEnMotRight    14
#define FwrdSpeedMotRight 26
#define RvrsSpeedMotRight 27

//enable: 14, L: 27, R:26

void setup() {
preferences.begin("myapp", false);
pinMode(FwrdEnMotRight ,   OUTPUT);
pinMode(FwrdSpeedMotRight, OUTPUT);
pinMode(RvrsSpeedMotRight, OUTPUT);


// configure LED PWM functionalitites
ledcSetup(pwmChannel_0, freq, resolution);
ledcSetup(pwmChannel_1, freq, resolution);


// attach the channel to the GPIO to be controlled
ledcAttachPin(FwrdSpeedMotRight, pwmChannel_0);
```

```
ledcAttachPin(RvrsSpeedMotRight, pwmChannel_1);
SerialMon.begin(115200);
delay(10);


Serial2.begin(19200, SERIAL_8N1, RxArdu, TxArdu);// serial2 pines 32 y 33
SerialMon.print("El valor de conexion es");
SerialMon.println(preferences.getUInt("counter", 0));
if (preferences.getUInt("counter", 0)==0) {
SerialMon.println("no hay nada ");
preferences.putUInt("counter", 1);
}

if (preferences.getUInt("counter", 0)==2) {
SerialMon.println("Se conecta con GPRS");
conectGprs();
mqtt.setClient(client);
}

if (preferences.getUInt("counter", 0)==1) {
SerialMon.println("Se conecta con Wifi");
conectWifi();
mqtt.setClient(espClient);
}

// MQTT Broker setup
mqtt.setServer(broker, 1883);
mqtt.setCallback(mqttCallback);


/*
```

```
float calValue_1; // Variable de calibracion celda 1
float calValue_2; // Variable de calibracion celda 2
calValue_1 = 761.80;//696.0; // calibrado el
calValue_2 = 620.64;//733.0; // calibrado el
calValue_3 = 620.64;//733.0; // calibrado el
LoadCell_1.begin();
LoadCell_2.begin();
LoadCell_3.begin();
long stabilisingtime = 2000; // Se neceita algunos segundos para realizar la tara
byte loadcell_1_rdy = 0;
byte loadcell_2_rdy = 0;
byte loadcell_3_rdy = 0;
while ((loadcell_1_rdy + loadcell_2_rdy+loadcell_3_rdy) < 3) { //corren ambos modulos simultanemente
if (!loadcell_1_rdy) loadcell_1_rdy = LoadCell_1.startMultiple(stabilisingtime);
if (!loadcell_2_rdy) loadcell_2_rdy = LoadCell_2.startMultiple(stabilisingtime);
if (!loadcell_3_rdy) loadcell_3_rdy = LoadCell_3.startMultiple(stabilisingtime);
}
LoadCell_1.setCalFactor(calValue_1); // user set calibration value (float)
LoadCell_2.setCalFactor(calValue_2); // user set calibration value (float)
LoadCell_3.setCalFactor(calValue_3); // user set calibration value (float)
Serial.println("Arranque de celdas y TARA compeltados");
*/
}

void loop() {

if (Serial2.available() > 0) {
deserializeJson(docSensoresRx, Serial2);
//serializeJson(docSensoresRx, Serial);
valTemp1 = docSensoresRx["temp1"];
valTemp2 =    docSensoresRx["temp2"];
```

```
valTemp3 = docSensoresRx["temp3"];
valTemp4 = docSensoresRx["temp4"];
valTempAmb1 = docSensoresRx["tempamb1"] ;
valTempAmb2 = docSensoresRx["tempamb2"] ;
valPresAmb1 = docSensoresRx["presamb1"] ;
valPresAmb2 = docSensoresRx["presamb2"] ;
valTdsAgua = docSensoresRx["tdsagua"];
valPh = docSensoresRx["ph"];
}
if (!mqtt.connected()) {

SerialMon.println("=== MQTT NOT CONNECTED ===");
// Reconnect every 10 seconds
uint32_t t = millis();
if (t - lastReconnectAttempt > 3000L) {
lastReconnectAttempt = t;
if (mqttConnect()) {
lastReconnectAttempt = 0;
}
cont++;
SerialMon.println(cont);
SerialMon.println("Se cumplieron los 3 seg");
if (cont > 4) {
SerialMon.println("Se cumplieron los 3 intentos de reconect");
if (preferences.getUInt("counter", 0)==2) {
SerialMon.println("conectado por wifi cambia a gprs");
delay(100);
preferences.putUInt("counter", 1);
delay(100);
} else {
if (preferences.getUInt("counter", 0)==1) {
```

```
SerialMon.println("conectado por gprs cambia a wifi");
delay(100);

preferences.putUInt("counter", 2);
delay(100);
}
}
SerialMon.println("conectado por gprs cambia a wifi");
preferences.putUInt("counter", 2);
delay(1000);
//mandar a reiniciar
//ESP.restart();
}
}
delay(100);
return;

}
else {
long now = millis();
if (now - lastMsg > 10000) {
lastMsg = now;
docTemp["temp1"] = valTemp1;
docTemp["temp2"] = valTemp2;
docTemp["temp3"] = valTemp3;
docTemp["temp4"] = valTemp4;


docPeso["peso1"] = random(100, 2000);
docPeso["peso2"] = random(100, 2000);
docPeso["peso3"] = random(100, 2000);
```

```
docPeso["peso4"] = random(100, 2000);

docBMP["tempamb1"] = valTempAmb1;
docBMP["tempamb2"] = valTempAmb2;
docBMP["presamb1"] = valPresAmb1;
docBMP["presamb2"] = valPresAmb2;

docTDS["tdsagua"] = valTdsAgua;

docPH["ph"] = valPh;



char buffer1[256];
char buffer2[256];
char buffer3[256];
char buffer4[16];
char buffer5[16];
//docTemp, docPeso, docBMP, docTDS, docPH
size_t n1 = serializeJson(docTemp, buffer1);
size_t n2 = serializeJson(docPeso, buffer2);
size_t n3 = serializeJson(docBMP, buffer3);
size_t n4 = serializeJson(docTDS, buffer4);
size_t n5 = serializeJson(docPH, buffer5);
//client.publish("outTopic", buffer, n);
mqtt.publish("hidroponia/nodo1/sensores/temperatura", buffer1, n1);
//mqtt.publish("hidroponia/nodo1/sensores/peso",        buffer2, n2);
mqtt.publish("hidroponia/nodo1/sensores/bmp",         buffer3, n3);
//mqtt.publish("hidroponia/nodo1/sensores/tds",         buffer4, n4);
//mqtt.publish("hidroponia/nodo1/sensores/ph",          buffer5, n5);
SerialMon.println("Los valores a enviar :");
serializeJson(docTemp, SerialMon);
```

```
serializeJson(docPeso, SerialMon);

serializeJson(docBMP, SerialMon);

//serializeJson(docTDS, SerialMon);

//serializeJson(docPH, SerialMon);

//Serial.print(tempString1); Serial.print(" ");Serial.print(tempString2); Serial.print("
");Serial.print(tempString3); Serial.print(" ");Serial.print(tempString4);

SerialMon.println("");

SerialMon.print("Conexión actual");

SerialMon.println(tipo);

SerialMon.print("Conexión actual grabada ");

SerialMon.println( preferences.getUInt("counter", 0));

}

//SerialMon.println("");

}



mqtt.loop();

}



void mqttCallback(char* topic, byte* message, unsigned int length) {


SerialMon.print("Message arrived on topic: ");

SerialMon.print(topic);

SerialMon.print(". Message: ");

if (String(topic) == "hidroponia/nodo1/conexion1") {

SerialMon.print("Conexion ");




} else {

deserializeJson(docDisparadores, (const byte*)message, length);
```

```
//Serial.println(docDisparadores);


SerialMon.println("");
if (docDisparadores["nombre"] == "modulo1") {
SerialMon.println("Recibo modulo1");
}
if (docDisparadores["nombre"] == "modulo2") {
SerialMon.println("Recibo modulo2");
}
if (docDisparadores["nombre"] == "motores") {
SerialMon.println("Recibo motores");
}
serializeJson(docDisparadores, Serial2);
serializeJson(docDisparadores, SerialMon);
SerialMon.println();
if(String(topic) == "hidroponia/nodo1/motores"){
if (docDisparadores["nombre"] == "motor4" && docDisparadores["valor"] == 1) {
int velMotor4 = 255;
digitalWrite(FwrdEnMotRight, HIGH);
ledcWrite(pwmChannel_0, velMotor4);          // FwrdSpeedMotRight
ledcWrite(pwmChannel_1, 0);          // FwrdSpeedMotLeft

}
if (docDisparadores["nombre"] == "motor4" && docDisparadores["valor"] == 0) {
int  velMotor4 = 0;
digitalWrite(FwrdEnMotRight, LOW);

//Serial.println("apagar  motor 1 ");
}

if (docDisparadores["nombre"] == "motores" && docDisparadores["valor"] == 1) {
```

```
int  velMotor4 = docDisparadores["vel4"];
digitalWrite(FwrdEnMotRight, HIGH);
ledcWrite(pwmChannel_0, velMotor4);          // FwrdSpeedMotRight
ledcWrite(pwmChannel_1, 0);          // FwrdSpeedMotLeft



//  Serial.println("apagar  motor 3 ");
}
if (docDisparadores["nombre"] == "motores" && docDisparadores["valor"] == 0) {
int  velMotor4 = 0;
digitalWrite(FwrdEnMotRight, LOW);



//  Serial.println("apagar  motor 3 ");
}

if (docDisparadores["conexion"] == "gprs") {
SerialMon.println("GPRS");
preferences.putUInt("counter", 2);

delay(1000);

SerialMon.println(  preferences.getUInt("counter", 0));
//mandar a reiniciar
ESP.restart();



//  Serial.println("apagar  motor 3 ");
}
```

```
if (docDisparadores["conexion"] == "wifi") {
SerialMon.println("wifi");
preferences.putUInt("counter", 1);

delay(1000);

SerialMon.println(  preferences.getUInt("counter", 0));
//mandar a reiniciar
ESP.restart();



//  Serial.println("apagar  motor 3 ");
}



}
}



}

bool mqttConnect() {
// Loop until we're reconnected
SerialMon.print("Connecting MQTT ");
SerialMon.print(broker);

boolean status = mqtt.connect(idClient, userMqtt, passMqtt);

if (status == false) {
SerialMon.println(" fail");
return false;
```

```
}
SerialMon.println(" success");
// mqtt.publish(topicInit, "GsmClientTest started");
mqtt.subscribe("hidroponia/nodo1/disparadores/#");
mqtt.subscribe("hidroponia/nodo1/motores/#");
mqtt.subscribe("hidroponia/nodo1/conexion/");
mqtt.subscribe("conexion");
return mqtt.connected();

}




void conectWifi() {
// Wifi connection parameters must be set before waiting for the network
SerialMon.print(F("Setting SSID/password..."));
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
SerialMon.print(".");

}
SerialMon.println("");
SerialMon.println("WiFi connected");
SerialMon.println("IP address: ");
SerialMon.println(WiFi.localIP());

}

void conectGprs() {
```

```
// Keep power when running from battery

tipo="Gsm";
// Set modem reset, enable, power pins
pinMode(MODEM_PWKEY, OUTPUT);
pinMode(MODEM_RST, OUTPUT);
pinMode(MODEM_POWER_ON, OUTPUT);
digitalWrite(MODEM_PWKEY, LOW);
digitalWrite(MODEM_RST, HIGH);
digitalWrite(MODEM_POWER_ON, HIGH);

// Set GSM module baud rate and UART pins
SerialAT.begin(115200, SERIAL_8N1, MODEM_RX, MODEM_TX);
delay(6000);

// Restart SIM800 module, it takes quite some time
// To skip it, call init() instead of restart()
SerialMon.println("Initializing modem...");
modem.restart();
// use modem.init() if you don't need the complete restart
String modemInfo = modem.getModemInfo();

// Unlock your SIM card with a PIN if needed
if (simPIN && modem.getSimStatus() != 3 ) {
modem.simUnlock(simPIN);
}

SerialMon.print(F("Connecting to GPRS "));
SerialMon.print(apn);
if (!modem.gprsConnect(apn, gprsUser, gprsPass)) {
```

```
SerialMon.println(" fail");

delay(5000);

return;

}

SerialMon.println(" nos conectamos a apn claro");


if (modem.isGprsConnected()) {

SerialMon.println("GPRS connected");

}


}
```

5.2. Controlador Fuzzy – Rasberry Pi

```
# -*- coding: utf-8 -*-

##Program to demonstrate the usage of threading.Timer() class within Python

import threading as th

import numpy as np

from matplotlib import pyplot as plt

from fuzzy import *

import paho.mqtt.client as mqtt # import the client1

import tkinter as tk

import RPi.GPIO as GPIO

from time import sleep

import json

import time


broker_address = "178.62.53.149"

broker_port = 1883

#topic = "led1"

mqtt_topics = ["hidroponia/nodo1/aut_off","hidroponia/nodo1/aut_on","hidroponia/nodo1/sensores/valor",
"hidroponia/nodo1/sensores/tds1", "hidroponia/nodo1/sensores/ph", "hidroponia/nodo1/sensores/nivel",
"hidroponia/nodo1/disparadores/modulosexpan", "mousex", "mousey", "led1", "ghi",
```

```
"hidroponia/nodo1/disparadores/modulosexpan/modulo1",
"hidroponia/nodo1/sensores","hidroponia/nodo1/motores","hidroponia/nodo1/sensores/temperatura","hidrop
onia/nodo1/sensores/bmp", "hidroponia/nodo1/sensores/ph"] # Change to multiple topics that suits your
needs.

Mensaje = False

vez=0

#time.sleep(10) # Paramos el hilo para recibir mensajes.

nivel=0

temp1=0


tempamb1=0

ph=0

PH1=0

GPIO21 = 21

GPIO20 = 20

tdsagua=0

sim_PH=0

Sal = 0

Sal_sim_deseada=0

PH_deseado=0

Sal_deseada=0

GPIO.setmode(GPIO.BCM)

GPIO.setup(GPIO21, GPIO.OUT)

GPIO.setup(GPIO20, GPIO.OUT)


master = tk.Tk()

master.title("GPIO Control")

master.geometry("800x600")

AP_defuzz=0

Nu_defuzz=0

GPIO21_state = True

GPIO20_State = True
```

```
rele1_state = True
rele2_state = True
rele3_state = True
rele4_state = True
rele5_state = True
rele6_state = True
motor1_state = True
motor2_state = True
motor3_state = True
motor4_state = True
automatico_state=True

w1 = tk.Scale(master, from_=14, to=1)
w1.set(10)
w1.place(x = 300, y = 190)
w2 = tk.Scale(master, from_=0, to=1500)
w2.set(23)
w2.place(x = 360, y =190)

w3 = tk.Scale(master, from_=14, to=1)
w3.set(10)
w3.place(x = 150, y = 190)
w4 = tk.Scale(master, from_=0, to=1500)
w4.set(23)
w4.place(x = 210, y =190)



def prnt_AP():
    global AP_defuzz, Nu_defuzz,valor_t_AP, message
    print("AP TEST \n")
    if AP_defuzz > 0:
```

```
      print("es Mayor")
      valor_t_AP =round(AP_defuzz,1)
      print(valor_t_AP)
      #message= '{ "nombre":"motor3",  "valor": 0}'
      #client.publish("hidroponia/nodo1/motores",message)#publish
    else:
      print("es Menor")
      print(AP_defuzz)
      valor_t_AP =round(AP_defuzz,1)
      print(valor_t_AP )
      #message= '{ "nombre":"motor4",  "valor": 0}'
      #client.publish("hidroponia/nodo1/motores",message)#publish

def prnt_Nu():
    global AP_defuzz, Nu_defuzz, valor_t_Nu
    print("NU TEST \n")
    if Nu_defuzz > 0:
      print("es mayor")
      valor_t_Nu =round(Nu_defuzz,1)
      #message= '{ "nombre":"motor1",  "valor": 0}'
      #client.publish("hidroponia/nodo1/motores",message)#publish
      print(valor_t_Nu)
      valor=round(Nu_defuzz,3)
      print(valor)
      print("DISMINUIR SALES")
    else:
      print("es menor")
      valor_t_Nu =round(Nu_defuzz,1)
      #message= '{ "nombre":"motor2",  "valor": 0}'
      #client.publish("hidroponia/nodo1/motores",message)#publish
      print(Nu_defuzz)
```

```
        print("DISMINUIR SALES")


def set_velocidad1():
    global velo1, ph,tdsagua,AP_defuzz, valor
    velo1=int(vel1.get())
    print(velo1)
    #print(type(PH))
    message= '{ "nombre":"motor1", "vel1": '
    message2='}'
    message_vel1=message+str(velo1)+message2
    client.publish("hidroponia/nodo1/motores/modulo1/",message_vel1)#publish

    return velo1

def set_velocidad2():
    global velo2, ph,tdsagua,AP_defuzz, valor
    velo2=int(vel2.get())
    print(velo2)
    #print(type(PH))
    message= '{ "nombre":"motor2", "vel2": '
    message2='}'
    message_vel2=message+str(velo2)+message2
    client.publish("hidroponia/nodo1/motores/modulo1/",message_vel2)#publish

    return velo2

def set_velocidad3():
    global velo3, ph,tdsagua,AP_defuzz, valor
    velo3=int(vel3.get())
    print(velo3)
    #print(type(PH))
```

```
    message= '{ "nombre":"motor3", "vel3": '
    message3='}'
    message_vel3=message+str(velo3)+message3
    client.publish("hidroponia/nodo1/motores/modulo1/",message_vel3)#publish

    return velo3
def set_velocidad4():
    global velo4, ph,tdsagua,AP_defuzz, valor
    velo4=int(vel4.get())
    print(velo4)
    #print(type(PH))
    message= '{ "nombre":"motor4", "vel4": '
    message4='}'
    message_vel4=message+str(velo4)+message4
    client.publish("hidroponia/nodo1/motores/modulo1/",message_vel4)#publish

    return velo4


def set_ph():
    global PH, ph,tdsagua,AP_defuzz, valor, PH_deseado
    PH=int(w1.get())
    print(PH)
    PH_deseado=int(w1.get())
    valor=round(AP_defuzz,3)
    print(valor)
    #print(type(PH))
    #Sal = 900  # Leer valor del sensor de salinidad
    #message= '{ "ph": "'
    #message2='"}'
    #message_ph=message+str(valor)+message2
```

```
    PH_des_label.config(text="PH Deseado: " +str(PH_deseado))
    #client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message_ph)#publish

    return PH_deseado
def set_tds():
    global Sal,Nu_defuzz,valor_t_AP,AP_defuzz,w1, Sal_deseada
    #valor_t_AP =round(AP_defuzz,1)*10
    #valor_t_AP=abs(valor_t_AP)
    #print(valor_t_AP)
    print("SET TDS")
    Sal_deseada=int(w2.get())
    print(Sal_deseada)
    Sal_des_label.config(text="Sal Deseada: " +str(Sal_deseada))
    #T1 = th.Timer(valor_t_AP, prnt_AP)
    #T1.start()
    #T2 = th.Timer(3.0, prnt_Nu)
    #T2.start()
    #message4= '{ "tds": "'
    #message5='"}'
    #valor=round(Nu_defuzz,3)
    #message_tds=message4+str(valor)+message5
    #client.publish("hidroponia/nodo1/set_tds/",Sal_deseada)#publish
    return Sal

def set_sim_ph():
    global sim_PH, ph,tdsagua,AP_defuzz, valor, PH_deseado
    sim_PH=int(w3.get())
    print(sim_PH)
    PH_deseado=int(w3.get())
    valor=round(AP_defuzz,3)
    print(valor)
```

```
    #print(type(PH))
    #Sal = 900  # Leer valor del sensor de salinidad
    #message= '{ "ph": "'
    #message2='"}'
    #message_ph=message+str(valor)+message2
    PH_SIM_label.config(text="PH_SIM: " +str(sim_PH))

    #client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message_ph)#publish

    return PH
def set_sim_tds():
    global Sal_sim_deseada,Nu_defuzz,valor_t_AP,AP_defuzz,w1
    #valor_t_AP =round(AP_defuzz,1)*10
    #valor_t_AP=abs(valor_t_AP)
    #print(valor_t_AP)
    print("SET TDS")
    Sal_sim_deseada=int(w4.get())
    print(Sal_sim_deseada)

    #Sal_SIM_label.config(text="Sal Sim Deseada: " +str(Sal_sim_deseada))
    #T1 = th.Timer(valor_t_AP, prnt_AP)
    #T1.start()
    #T2 = th.Timer(3.0, prnt_Nu)
    #T2.start()
    #message4= '{ "tds": "'
    #message5='"}'
    #valor=round(Nu_defuzz,3)
    #message_tds=message4+str(valor)+message5

    #client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message_tds)#publish
    return Sal
```

```
def leerSensores():
    #global PH
    global ph,tdsagua, t_actual,sim_PH,Sal_sim_deseada
    #PH=w1.get()
    Sal_sim_deseada=w4.get()
    #value_PH = PH  # Leer valor del sensor de PH
    #value_Sal = Sal  # Leer valor del sensor de salinidad
#     sim_PH=0
    #Sal_sim_deseada=0
    #PH = 8  # Leer valor del sensor de PH
    #tdsagua_label = tk.Label(master,text =tdsagua)
    #tdsagua_label.place(x = 450,y = 160)
    #tdsagua_label.config(text="Sales disueltas: " +str(tdsagua))


    PH=sim_PH
    #Sal = 100  # Leer valor del sensor de salinidad
    #Sal=Sal_sim_deseada
    Sal = int(tdsagua)  # Leer valor del sensor de salinidad
    #print(type(PH))
    #print(type(Sal))

    return PH, Sal

    #return ph, tdsagua

# Función para setear el tiempo de apertura de las válvulas

def setValves(AP, Nu):
```

```
    # Setear el tiempo de apertura del agua potable
    # Setear el tiempo de apertura del nutriente
    print(AP, Nu)

# Variables linguisticas de entrada
# Para el error de PH
# Sensor de PH puede medir de 1 a 14
# Error de PH: -13 a 13
# Paso: 1.7
# Universo de discurso para el error de PH

PH_e = np.linspace(-13, 13, 100)


# Funciones de membresía para el error de PH
PH_ENG = [trapmf, [-13, -13, -3.4, -1.7]]
PH_ENP = [trimf, [-3.4, -1.7, 0.0]]
PH_EC = [trimf, [-1.7, 0.0, 1.7]]
PH_EPP = [trimf, [0.0, 1.7, 3.4]]
PH_EPG = [trapmf, [1.7, 3.4, 13, 13]]

# Para el error de salinidad
# Sensor de salinidad puede medir de 0 a 2000 ppm
# Error de PH: -2000 a 2000 ppm
# Paso: 575

# Universo de discurso para el error de salinidad
#Sal_e = np.linspace(-2000, 2000, 100)
#Sal_e = np.linspace(-200, 200, 100)
# Funciones de membresía para el error de salinidad
#Sal_ENG = [trapmf, [-2000, -2000, -1150, -575]]
```

```
#Sal_ENP = [trimf, [-1150, -575, 0.0]]

#Sal_EC = [trimf, [-575, 0.0, 575]]

#Sal_EPP = [trimf, [0.0, 575, 1150]]

#Sal_EPG = [trapmf, [575, 1150, 2000, 2000]]


# Universo de discurso para el error de salinidad

Sal_e = np.linspace(-1000, 1000, 50)

#Sal_e = np.linspace(-200, 200, 100)

# Funciones de membresía para el error de salinidad

#Sal_ENG = [trapmf, [-1000, -1000, -750, -250]]

#Sal_ENP = [trimf, [-750, -250, 0.0]]


#Sal_EC = [trimf, [-400, 0.0, 400]]

#Sal_EPP = [trimf, [0.0, 250, 750]]

#Sal_EPG = [trapmf, [250, 750, 1000, 1000]]



Sal_ENG = [trapmf, [-1000, -1000, -750, -250]]

Sal_ENP = [trimf, [-250, -100, 0.0]]

#Sal_EC = [trimf, [-250, 0.0, 250]]

Sal_EC = [trimf, [-50, 0.0, 50]]

Sal_EPP = [trimf, [0.0, 100, 250]]

Sal_EPG = [trapmf, [250, 750, 1000, 1000]]



#Sal_ENG = [trapmf, [-200, -200, -115, -57]]

#Sal_ENP = [trimf, [-115, -57, 0.0]]

#Sal_EC = [trimf, [-57, 0.0, 57]]

#Sal_EPP = [trimf, [0.0, 57, 115]]

#Sal_EPG = [trapmf, [57, 115, 200, 200]]
```

```
# Variables linguisticas de salida

# Para el tiempo de suministro de nutrientes
# Tiempo seteado de 0 a 10s
# Error tiempo: -10 a 10 s
# Paso: 3

# Universo de discurso para tiempo de suministro de nutrientes
#Nu_t = np.linspace(-10, 10, 100)

# Funciones de membresía para tiempo de suministro de nutrientes
#Nu_ME = [trapmf, [-10, -10, -6, -3]]
#Nu_PE = [trimf, [-6, -3, 0]]
#Nu_NE = [trimf, [-3, 0, 3]]
#Nu_PT = [trimf, [0, 3, 6]]
#Nu_MT = [trapmf, [3, 6, 10, 10]]

Nu_t = np.linspace(-255, 255, 100)
#255/7=36=40  80
Nu_ME = [trapmf, [-255, -255, -220, -180]]
Nu_PE = [trimf, [-180, -160, -120]]
Nu_NE = [trimf, [-120, 0, 120]]
Nu_PT = [trimf, [120, 160, 180]]
Nu_MT = [trapmf, [180, 220, 255, 255]]



# Para el tiempo de suministro de agua potable
```

```
# Tiempo seteado de 0 a 10s
# Error tiempo: -10 a 10 s
# Paso: 3

# Universo de discurso para tiempo de suministro de agua potable
AP_t = np.linspace(-10, 10, 100)

# Funciones de membresía para suministro de agua potable
AP_ME = [trapmf, [-10, -10, -6, -3]]
AP_PE = [trimf, [-6, -3, 0]]
AP_NE = [trimf, [-3, 0, 3]]
AP_PT = [trimf, [0, 3, 6]]
AP_MT = [trapmf, [3, 6, 10, 10]]

# RELACIONES:

PH = [PH_ENG, PH_ENP, PH_EC, PH_EPP, PH_EPG]
Sal = [Sal_ENG, Sal_ENP, Sal_EC, Sal_EPP, Sal_EPG]
Nu = [Nu_ME, Nu_PE, Nu_NE, Nu_PT, Nu_MT]
AP = [AP_ME, AP_PE, AP_NE, AP_PT, AP_MT]



"""
# Curvas de control para defuzzificación:
out_centroid = np.zeros(PH_e.size)

for i in range(PH_e.size):
    x0 = float(PH_e[i])
    Bp = fuzz(x0, Nu_t, PH, Nu)
    out_centroid[i] = defuzz(Nu_t, Bp, 'centroid')
```

```
plt.plot(PH_e, out_centroid, label="centroid")
plt.legend()
plt.xlabel('error')
plt.ylabel('out')
plt.show()
"""

# Parámetros de operación
t_actual = 0   # tiempo inicial
t_final = 1    # tiempo final
#dt = 0.001     # diferencial de tiempo
dt = 0.1     # diferencial de tiempo
# Valores iniciales

#PH_deseado = 5.8
#Sal_deseada = 200
#PH_deseado = 9

#Sal_deseada = 400

#PH_des_label = tk.Label(master,text ="PH")
#PH_des_label.place(x = 620,y = 220)
Sal_des_label = tk.Label(master,text ="Sal")
Sal_des_label.place(x = 620,y = 250)
#PH_des_label.config(text="PH Deseado: " +str(PH_deseado))
Sal_des_label.config(text="Sal Deseada: " +str(Sal_deseada))

PH_actual, Sal_actual = leerSensores()

t = [t_actual]
Ph_f = [PH_actual]
```

```
Sal_f = [Sal_actual]



def FuzzyOn():
    global PH_actual, Sal_actual
    global PH_e, PH_deseado, PH, AP
    global Sal_e, Sal_deseada, AP_t
    global Sal_fuzz, Nu_t, Sal, Nu_t,Nu
    global AP_defuzz,Nu_defuzz, t_actual
    global t_final, client, userdata, message, automatico_state
    global ph, time,tdsagua,GPIO21_state, vez
    global Sal,Nu_defuzz,valor_t_AP,AP_defuzz,valor_t_Nu
    #while (True):
    print("Fuzzy Apagado ")
    while (automatico_state == True):
        print("Fuzzy Encendido ")
        vez=vez+1
        print(vez)
        client.publish("hidroponia/nodo1/set_tds",Sal_deseada)#publish


        # Obtenemos la posición actual y el error


        PH_actual, Sal_actual = leerSensores()
        #PH_actual=PH
        #Sal_actual=Sal


        PH_e = PH_deseado - PH_actual
        Sal_e = Sal_deseada - Sal_actual
        print("Diferencia Setpoint")
        print(Sal_e)
        # Fuzzificación
```

```
PH_fuzz = fuzz(PH_e, AP_t, PH, AP)
Sal_fuzz = fuzz(Sal_e, Nu_t, Sal, Nu)

# Defuzzificación
AP_defuzz = defuzz(AP_t, PH_fuzz, 'centroid')
Nu_defuzz = defuzz(Nu_t, Sal_fuzz, 'centroid')

setValves(AP_defuzz, Nu_defuzz)

# guardamos los datos actuales para graficar
t.append(t_actual)
Ph_f.append(PH_actual)
Sal_f.append(Sal_actual)

# actualizamos el vector de tiempo para el siguiente paso
t_actual += dt
print(t_actual)

# y verificamos las condiciones de salida
if t_actual >= t_final:
    t_actual=0
    #message= '{ "nombre":"motor2",  "valor": 1}'
    #client.publish("hidroponia/nodo1/motores",message)#publish
    #print("Slor encendido")
    break
#time.sleep(3)
Nu_defuzz=Nu_defuzz
if Nu_defuzz > 250:
    Nu_defuzz=250
    client.publish("hidroponia/nodo1/estado","Aumentando Sales")#publis
if Nu_defuzz < -250:
```

```
    Nu_defuzz=-250
if Nu_defuzz > 112:
    print("es mayor")
    valor_t_Nu =round(Nu_defuzz,1)
    print(valor_t_Nu)
    valor_t_Nu=int(valor_t_Nu)
    #message= '{ "nombre":"motor1",  "valor": 1}'
    #client.publish("hidroponia/nodo1/motores",message)#publish
    motor4_off();
    message= '{ "nombre":"motor2", "vel2": '
    message2='}'
    message_vel1=message+str(0)+message2
    client.publish("hidroponia/nodo1/motores/modulo1/",message_vel1)#publish
    valor_t_Nu=int(valor_t_Nu)

    client.publish("hidroponia/nodo1/estado","Aumentando Sales")#publish
    MAS_TDS_T_label = tk.Label(master,text ="Aumentando Sales")
    MAS_TDS_T_label.place(x = 450,y = 250)
    MAS_TDS_T_label.config(text="Aumentando Sales")
    message= '{ "nombre":"motor1", "vel1": '
    message2='}'
    message_vel1=message+str(valor_t_Nu)+message2
    client.publish("hidroponia/nodo1/motores/modulo1/",message_vel1)#publish

if Nu_defuzz < -120:
    motor4_off();
    print("es menor")
    valor_t_Nu =round(Nu_defuzz,1)
    #message= '{ "nombre":"motor2",  "valor": 1}'
    #client.publish("hidroponia/nodo1/motores",message)#publish
    MAS_TDS_T_label = tk.Label(master,text ="Disminuyendo Sales")
```

```
    MAS_TDS_T_label.place(x = 450,y = 250)
    message= 'Disminuyendo Sales'
    client.publish("hidroponia/nodo1/estado",message)#publish
    MAS_TDS_T_label.config(text="Disminuyendo Sales")
    message= '{ "nombre":"motor1", "vel1": '
    message2='}'
    message_vel1=message+str(0)+message2
    client.publish("hidroponia/nodo1/motores/modulo1/",message_vel1)#publish




    message= '{ "nombre":"motor2", "vel2": '
    message2='}'
    valor_t_Nu=-int(valor_t_Nu)
    message_vel1=message+str(valor_t_Nu)+message2
    client.publish("hidroponia/nodo1/motores/modulo1/",message_vel1)#publish

if Nu_defuzz > -112 and Nu_defuzz < 112:
    #T4 = th.Timer(20,motor4_button)
    #T4.start()
    motor4_on();
    print("es menor")
    valor_t_Nu =round(Nu_defuzz,1)



    message= '{ "nombre":"motor1", "vel1": '
    message2='}'
    message_vel1=message+str(0)+message2
    client.publish("hidroponia/nodo1/motores/modulo1/",message_vel1)#publish
```

```
    message= '{ "nombre":"motor2", "vel2": '
    message2='}'
    message_vel1=message+str(0)+message2
    client.publish("hidroponia/nodo1/motores/modulo1/",message_vel1)#publish
    Sal_T_label.config(text="T_ACT_TDS: " +"Se mantiene")

    #message= '{ "nombre":"motor2",  "valor": 1}'
    #client.publish("hidroponia/nodo1/motores",message)#publish
    MAS_TDS_T_label = tk.Label(master,text ="Manteniendo Sales")
    MAS_TDS_T_label.place(x = 450,y = 250)
    MAS_TDS_T_label.config(text="Manteniendo Sales")
    message= 'Manteniendo Sales'
    client.publish("hidroponia/nodo1/estado",message)#publish


else:
    print("es menor")
    valor_t_Nu =round(Nu_defuzz,1)
    #message= '{ "nombre":"motor4",  "valor": 1}'
    #client.publish("hidroponia/nodo1/motores",message)#publish
    MAS_TDS_T_label = tk.Label(master,text ="Disminuyendo Sales")
    MAS_TDS_T_label.place(x = 450,y = 280)
    MAS_TDS_T_label.config(text="Disminuyendo Sales")
    message= 'Disminuyendo Sales'
    #client.publish("hidroponia/nodo1/estado",message)#publish
```

```
        valor_t_AP =round(AP_defuzz,1)
        #valor_t_AP=abs(valor_t_AP)
        valor_t_AP_abs=abs(valor_t_AP)
        valor_t_Nu =round(Nu_defuzz,1)
        valor_t_Nu_abs=abs(valor_t_Nu)
        PH_T_label.config(text="T_ACT_PH: " +str(valor_t_AP))
        Sal_T_label.config(text="T_ACT_TDS: " +str(valor_t_Nu))

        #valor_t_Nu =round(Nu_defuzz,1)*10
        #print(valor_t_Nu)
        #T2 = th.Timer(valor_t_Nu_abs, prnt_Nu)
        #T2.start()
        #T1 = th.Timer(valor_t_AP_abs, prnt_AP)
        #T1.start()
        #message= '{ "ph": "'
        #message_ph=message+PH_actual+'"out1", "valor": 1}'
        #client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message,PH_actual)#publish

def automatico():
    global automatico_state,Sal_deseada

    if automatico_state == True:
        automatico_state = False
        message= '{ "nombre":"motores",  "valor": 0}'
        client.publish("hidroponia/nodo1/motores",message)#publish
        ONlabel_aut = tk.Label(master, text="Turned ON", fg="green")
        client.publish("hidroponia/nodo1/disparadores/automat","Apagado")#publish
        print("APAGADO RELE1")
        ONlabel_aut.grid(row=13, column=1)
    else:
        automatico_state = True
```

```
        message= '{ "nombre":"motores",  "valor": 1}'
        client.publish("hidroponia/nodo1/motores",message)#publish
        client.publish("hidroponia/nodo1/disparadores/automat","Encendido")#publish
        ONlabel_aut = tk.Label(master, text="Turned OFF", fg="red")
        ONlabel_aut.grid(row=13, column=1)
        print("ENCENDIDO RELE1")

def GPIO21button():
    global GPIO21_state, GPIO20_State,automatico_state
    if GPIO21_state == True:



        GPIO21_state = False
        ONlabel = tk.Label(master, text="Turned ON", fg="green")
        #client.publish("values/","12,15,17")#publish
        print("OFF")
        ONlabel.grid(row=0, column=1)
    else:

        GPIO21_state = True

        ONlabel = tk.Label(master, text="Turned OFF", fg="red")
        ONlabel.grid(row=0, column=1)
        print("Owwfwf")



def nivela():
    global nivel,Sal_deseada,tdsagua
    client.publish("hidroponia/nodo1/set_tds",Sal_deseada)#publish
    if tdsagua > 250 and tdsagua < 1000 and automatico_state == True:
```

```
        message= '{ "nombre":"motores",  "valor": 1}'
        client.publish("hidroponia/nodo1/motores",message)#publish
        message= 'TDS En Rango'
        client.publish("hidroponia/nodo1/error",message)#publish
    if tdsagua<250 and automatico_state == True:

        message= '{ "nombre":"motores",  "valor": 0}'
        client.publish("hidroponia/nodo1/motores",message)#publish
        message= 'Error de lectura en TDS'
        client.publish("hidroponia/nodo1/error",message)#publish
        automatico()
    if nivel==1:

            #Testbutton3.config(bg="red")
        message= '{ "nombre":"motor3",  "valor": 1}'
        client.publish("hidroponia/nodo1/motores",message)#publish
        client.publish("hidroponia/nodo1/motor3","Descargando")#publish
            #Testbutton4.config(bg="red")
            #message= '{ "nombre":"motor4",  "valor": 1}'
            #client.publish("hidroponia/nodo1/motores",message)#publish
        print("nivel1")
    if nivel==0:

            #Testbutton4.config(bg="blue")
        message= '{ "nombre":"motor3",  "valor": 0}'
        client.publish("hidroponia/nodo1/motores",message)#publish
        client.publish("hidroponia/nodo1/motor3","En rango")#publish
        print("nivel0")
def GPIO20button():
    global GPIO20_State, nivel, client, userdata, message
```

```
    if GPIO20_State == True:

        OFFlabel1 = tk.Label(master, text="Turned ON", fg="green")
        OFFlabel1.grid(row=1, column=1)
        #FuzzyOn()
        #on_message(client, userdata, message)
        #on_message()
        #time.sleep(5)

            #Testbutton3.config(bg="blue")
            #message= '{ "nombre":Mmotor3",  "valor": 0}'
            #client.publish("hidroponia/nodo1/motores",message)#publish
         #tdsagua_label.config(text="Sales disueltas: " +str(nivel1))

    else:

        OFFlabel1 = tk.Label(master, text="Turned OFF", fg="red")
        OFFlabel1.grid(row=1, column=1)



def rele1_button():
            global rele1_state
            if rele1_state == True:
                        rele1_state = False
                        message= '{ "nombre":"modulo1", "salida": "out1", "valor": 1}'
                        ONlabel_rele1 = tk.Label(master, text="Turned ON", fg="green")

            client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
                        print("APAGADO RELE1")
                        ONlabel_rele1.grid(row=3, column=1)
            else:
```

```
                rele1_state = True
                message= '{ "nombre":"modulo1", "salida": "out1", "valor": 0}'

        client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
                ONlabel_rele1 = tk.Label(master, text="Turned OFF", fg="red")
                ONlabel_rele1.grid(row=3, column=1)
                print("ENCENDIDO RELE1")


def rele2_button():
        global rele2_state
        if rele2_state == True:
                rele2_state = False
                message= '{ "nombre":"modulo1", "salida": "out2", "valor": 1}'
                ONlabel_rele2 = tk.Label(master, text="Turned ON", fg="green")

        client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
                print("APAGADO RELE2")
                ONlabel_rele2.grid(row=4, column=1)
        else:
                rele2_state = True
                message= '{ "nombre":"modulo1", "salida": "out2", "valor": 0}'

        client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
                ONlabel_rele2 = tk.Label(master, text="Turned OFF", fg="red")
                ONlabel_rele2.grid(row=4, column=1)
                print("ENCENDIDO RELE2")



def rele3_button():
        global rele3_state
        if rele3_state == True:
                rele3_state = False
                message= '{ "nombre":"modulo1", "salida": "out3", "valor": 1}'
```

```
                ONlabel_rele3 = tk.Label(master, text="Turned ON", fg="green")

        client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
                print("APAGADO RELE3")
                ONlabel_rele3.grid(row=5, column=1)
        else:
                rele3_state = True
                message= '{ "nombre":"modulo1", "salida": "out3", "valor": 0}'

        client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
                ONlabel_rele3 = tk.Label(master, text="Turned OFF", fg="red")
                ONlabel_rele3.grid(row=5, column=1)
                print("ENCENDIDO RELE3")


def rele4_button():
        global rele4_state
        if rele4_state == True:
                rele4_state = False
                message= '{ "nombre":"modulo1", "salida": "out4", "valor": 1}'
                ONlabel_rele4 = tk.Label(master, text="Turned ON", fg="green")

        client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
                print("APAGADO RELE 4")
                ONlabel_rele4.grid(row=6, column=1)
        else:
                rele4_state = True
                message= '{ "nombre":"modulo1", "salida": "out4", "valor": 0}'

        client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
                ONlabel_rele4 = tk.Label(master, text="Turned OFF", fg="red")
                ONlabel_rele4.grid(row=6, column=1)
                print("ENCENDIDO RELE 4")
```

```
def rele5_button():
	global rele5_state
	if rele5_state == True:
		rele5_state = False
		message= '{ "nombre":"modulo1", "salida": "out5", "valor": 1}'
		ONlabel_rele5 = tk.Label(master, text="Turned ON", fg="green")

	client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
		print("rele5")
		time.sleep(3)
		message= '{ "nombre":"modulo1", "salida": "out4", "valor": 1}'

	client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
		print("rele4")
		time.sleep(3)
		message= '{ "nombre":"modulo1", "salida": "out3", "valor": 1}'

	client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
		print("rele3")
		time.sleep(3)
		message= '{ "nombre":"modulo1", "salida": "out2", "valor": 1}'

	client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
		print("rele2")
		time.sleep(3)
		message= '{ "nombre":"modulo1", "salida": "out1", "valor": 1}'

	client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
		print("rele1")
		ONlabel_rele5 = tk.Label(master, text="Turned ON", fg="green")
		print("APAGADO RELE5")
		ONlabel_rele5.grid(row=7, column=1)
	else:
		rele5_state = True
```

```
            message= '{ "nombre":"modulo1", "salida": "out5", "valor": 0}'

        client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
            print("rele5-0")
            time.sleep(2)
            message= '{ "nombre":"modulo1", "salida": "out4", "valor": 0}'

        client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
            print("rele4-0")
            time.sleep(2)
            message= '{ "nombre":"modulo1", "salida": "out3", "valor": 0}'

        client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
            print("rele3-0")
            time.sleep(2)
            message= '{ "nombre":"modulo1", "salida": "out2", "valor": 0}'

        client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
            print("rele2-0")
            time.sleep(2)
            message= '{ "nombre":"modulo1", "salida": "out1", "valor": 0}'

        client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
            print("rele1-0")
            time.sleep(2)



            ONlabel_rele5 = tk.Label(master, text="Turned OFF", fg="red")
            ONlabel_rele5.grid(row=7, column=1)
            print("ENCENDIDO RELE4")



def rele6_button():
        global rele6_state
```

```
        if rele6_state == True:
                rele6_state = False
                message= '{ "nombre":"modulo1", "salida": "out6", "valor": 1}'
                ONlabel_rele6 = tk.Label(master, text="Turned ON", fg="green")

        client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
                print("APAGADO RELE 6")
                ONlabel_rele6.grid(row=8, column=1)
        else:
                rele6_state = True
                message= '{ "nombre":"modulo1", "salida": "out6", "valor": 0}'

        client.publish("hidroponia/nodo1/disparadores/modulosexpan/modulo1/",message)#publish
                ONlabel_rele6 = tk.Label(master, text="Turned OFF", fg="red")
                ONlabel_rele6.grid(row=8, column=1)
                print("ENCENDIDO RELE4")


def motor1_button():
        global motor1_state
        if motor1_state == True:
                motor1_state = False
                message= '{ "nombre":"motor1",  "valor": 1}'
                ONlabel_motor1 = tk.Label(master, text="Turned ON", fg="green")
                client.publish("hidroponia/nodo1/motores",message)#publish
                print("APAGADO MOTOR 1")
                ONlabel_motor1.grid(row=9, column=1)
        else:
                motor1_state = True
                message= '{ "nombre":"motor1",  "valor": 0}'
                client.publish("hidroponia/nodo1/motores",message)#publish
                ONlabel_motor1 = tk.Label(master, text="Turned OFF", fg="red")
                ONlabel_motor1.grid(row=9, column=1)
```

```
                print("ENCENDIDO MOTOR 1")


def motor2_button():
        global motor2_state
        if motor2_state == True:
                motor2_state = False
                message= '{ "nombre":"motor2",  "valor": 0}'
                ONlabel_motor2 = tk.Label(master, text="Turned ON", fg="green")
                client.publish("hidroponia/nodo1/motores",message)#publish
                print("APAGADO MOTOR 2")
                ONlabel_motor2.grid(row=10, column=1)
        else:
                motor2_state = True
                message= '{ "nombre":"motor2",  "valor": 1}'
                client.publish("hidroponia/nodo1/motores",message)#publish
                ONlabel_motor2 = tk.Label(master, text="Turned OFF", fg="red")
                ONlabel_motor2.grid(row=10, column=1)
                print("ENCENDIDO MOTOR 2")


def motor3_button():
        global motor3_state
        if motor3_state == True:
                motor3_state = False
                message= '{ "nombre":"motor3",  "valor": 0}'
                ONlabel_motor3 = tk.Label(master, text="Turned ON", fg="green")
                client.publish("hidroponia/nodo1/motores",message)#publish
                print("APAGADO MOTOR 3")
                ONlabel_motor3.grid(row=11, column=1)
        else:
                motor3_state = True
                message= '{ "nombre":"motor3",  "valor": 1}'
```

```
                client.publish("hidroponia/nodo1/motores",message)#publish
                ONlabel_motor3 = tk.Label(master, text="Turned OFF", fg="red")
                ONlabel_motor3.grid(row=11, column=1)
                print("ENCENDIDO MOTOR1")

def motor4_button():
        global motor4_state
        if motor4_state == True:
                motor4_state = False
                message= '{ "nombre":"motor4",  "valor": 0}'
                ONlabel_motor4 = tk.Label(master, text="Turned ON", fg="green")
                client.publish("hidroponia/nodo1/motores",message)#publish
                client.publish("hidroponia/nodo1/motor4","Espera Set-Point")#publish
                print("APAGADO MOTOR 4")
                ONlabel_motor4.grid(row=12, column=1)
        else:
                motor4_state = True
                message= '{ "nombre":"motor4",  "valor": 1}'
                client.publish("hidroponia/nodo1/motores",message)#publish
                client.publish("hidroponia/nodo1/motor4","Recirculando")#publish
                ONlabel_motor4 = tk.Label(master, text="Turned OFF", fg="red")
                ONlabel_motor4.grid(row=12, column=1)
                print("ENCENDIDO MOTOR 4")

def motor4_on():
                message= '{ "nombre":"motor4",  "valor": 1}'
                client.publish("hidroponia/nodo1/motores",message)#publish
                client.publish("hidroponia/nodo1/motor4","Recirculando")#publish

def motor4_off():
                message= '{ "nombre":"motor4",  "valor": 0}'
```

```
                ONlabel_motor4 = tk.Label(master, text="Turned ON", fg="green")
                client.publish("hidroponia/nodo1/motores",message)#publish
                client.publish("hidroponia/nodo1/motor4","Espera Set-Point")#publish



def hab_motor():
        #global motor4_state
        message= '{ "nombre":"motores",  "valor": 1}'
        client.publish("hidroponia/nodo1/motores",message)#publish
        Hab_motor_button = tk.Label(master, text="HAB", fg="blue")
        Hab_motor_button.grid(row=15, column=1)
        print("MOTORES HABILITADOS")

def deshab_motor():
        #global motor4_state
        message= '{ "nombre":"motores",  "valor": 0}'
        client.publish("hidroponia/nodo1/motores",message)#publish
        Hab_motor_button = tk.Label(master, text="HAB", fg="blue")
        Hab_motor_button.grid(row=16, column=1)
        print("MOTORES DESAHABILITADOS")

def on_message(client, userdata, message):
    global Mensaje
    global consigneX
    global consigneY, y, Testbutton,automatico_state
    global tdsagua,ph, nivel, Sal_deseada,GPIO21_state,GPIO20_State

    mensaje=str(message.payload.decode("utf-8"))
    print(message.topic)
        #('Mensaje recibido=', '{ "nombre":"modulo1", "salida": "out1", "valor": 1 }')
```

```
        #hidroponia/nodo1/disparadores/modulosexpan/modulo1


if mensaje =="on":
    startBalanceBall == False
    Mensaje = True
    print("apagado")
    BStartBalance["text"] = "Desconectar"
if mensaje =="off":
    startBalanceBall == False
    Mensaje = False
    print("encendido")
    BStartBalance["text"] = "Desconectar"
if message.topic=="hidroponia/nodo1/disparadores/modulosexpan/modulo1":
    print("Mensaje recibido=", str(message.payload.decode("utf-8")))
    y = json.loads(mensaje)
    print(y["valor"])
    #print(type(y["valor"]))

    if y["salida"]=="out1":
        if y["valor"]==1:
            Testbutton1.config(bg="red")
        if y["valor"]==0:
            Testbutton1.config(bg="blue")

    if y["salida"]=="out2":
        if y["valor"]==1:
            Testbutton2.config(bg="red")
        if y["valor"]==0:
            Testbutton2.config(bg="blue")

    if y["salida"]=="out3":
```

```
            if y["valor"]==1:
                Testbutton3.config(bg="red")
            if y["valor"]==0:
                Testbutton3.config(bg="blue")

        if y["salida"]=="out4":
            if y["valor"]==1:
                Testbutton4.config(bg="red")
            if y["valor"]==0:
                Testbutton4.config(bg="blue")

        if y["salida"]=="out5":
            if y["valor"]==1:
                Testbutton5.config(bg="red")
            if y["valor"]==0:
                Testbutton5.config(bg="blue")

        if y["salida"]=="out6":
            if y["valor"]==1:
                Testbutton6.config(bg="red")
            if y["valor"]==0:
                Testbutton6.config(bg="blue")

    if message.topic=="hidroponia/nodo1/motores":
        print("Mensaje recibido=", str(message.payload.decode("utf-8")))
        y = json.loads(mensaje)
        print(y["valor"])
        #print(type(y["valor"]))

        if y["nombre"]=="motor1":
            if y["valor"]==1:
```

```
            Motorbutton1.config(bg="red")
        if y["valor"]==0:
            Motorbutton1.config(bg="blue")

    if y["nombre"]=="motor2":
        if y["valor"]==1:
            Motorbutton2.config(bg="red")
        if y["valor"]==0:
            Motorbutton2.config(bg="blue")
    if y["nombre"]=="motor3":
        if y["valor"]==1:
            Motorbutton3.config(bg="red")
        if y["valor"]==0:
            Motorbutton3.config(bg="blue")
    if y["nombre"]=="motor4":
        if y["valor"]==1:
            Motorbutton4.config(bg="red")
        if y["valor"]==0:
            Motorbutton4.config(bg="blue")



    #print(type(y["valor"]))



    #consigneX=int(message.payload.decode("utf-8"))
if message.topic=="hidroponia/nodo1/sensores/temperatura":
    print("Mensaje recibido=", str(message.payload.decode("utf-8")))
    #('Mensaje recibido=', '{"temp1":17,"temp2":19,"temp3":17,"temp4":20}')
    y = json.loads(mensaje)
    temp1= y["temp1"]
    temp2= y["temp2"]
```

```
    temp3= y["temp3"]
    temp4= y["temp4"]
    temp1_label.config(text="Temperatura 1: " +str(temp1))
    temp2_label.config(text="Temperatura 2: " +str(temp2))
    temp3_label.config(text="Temperatura 3: " +str(temp3))
    temp4_label.config(text="Temperatura 4: " +str(temp4))


if message.topic=="hidroponia/nodo1/sensores/bmp":
    print("Mensaje recibido=", str(message.payload.decode("utf-8")))
    # ('Mensaje recibido=', '{"tempamb1":10,"tempamb2":18,"presamb1":15,"presamb2":16}')
    y = json.loads(mensaje)
    tempamb1= y["tempamb1"]
    tempamb2= y["tempamb2"]
    #presamb1= y["presamb1"]
    #presamb2= y["presamb2"]
    tempamb1_label.config(text="Temperatura Amb. 1: " +str(tempamb1))
    tempamb2_label.config(text="Temperatura Amb.2: " +str(tempamb2))
    #presamb1_label.config(text="Pres. Amb. 1: " +str(presamb1))
    #presamb2_label.config(text="Pres. Amb.2: " +str(presamb2))

if message.topic=="hidroponia/nodo1/sensores/tds1":
    print("Mensaje recibido=", str(message.payload.decode("utf-8")))
    print("Llego TDS")
    # hidroponia/nodo1/sensores/tds('Mensaje recibido=', '{"tdsagua":29}')
    T1 = th.Timer(30,FuzzyOn)
    T1.start()
    #T1.join()
    T2 = th.Timer(5,nivela)
    T2.start()
    #y = json.loads(mensaje)
```

```
        #tdsagua= y["tdsagua"]
        tdsagua = json.loads(mensaje)
        #tdsagua=message.payload.decode("utf-8")
        tdsagua_label.config(text="Sales disueltas: " +str(tdsagua))


if message.topic=="hidroponia/nodo1/sensores/valor":
        print("Mensaje recibido=", str(message.payload.decode("utf-8")))
        print("Llego TDS_deseado")
        # hidroponia/nodo1/sensores/tds('Mensaje recibido=', '{"tdsagua":29}')

        #y = json.loads(mensaje)
        #tdsagua= y["tdsagua"]
        Sal_deseada = json.loads(mensaje)
        client.publish("hidroponia/nodo1/set_tds",Sal_deseada)#publish
        Sal_des_label.config(text="Sal Deseada: " +str(Sal_deseada))
        #tdsagua=message.payload.decode("utf-8")



if message.topic=="hidroponia/nodo1/sensores/ph":
        print("Mensaje recibido=", str(message.payload.decode("utf-8")))
        # hidroponia/nodo1/sensores/tds('Mensaje recibido=', '{"tdsagua":29}')
        ph=message.payload.decode("utf-8")
        #y = json.loads(mensaje)
        #ph= y["ph"]
        ph_label.config(text=" PH: " +str(ph))
if message.topic=="hidroponia/nodo1/sensores/nivel":
        print("Mensaje recibido=", str(message.payload.decode("utf-8")))
        # hidroponia/nodo1/sensores/tds('Mensaje recibido=', '{"tdsagua":29}')
```

```
        y = json.loads(mensaje)
        #ph= y["ph"]
        nivel= int(y)
        nivel_label.config(text=" Nivel: " +str(nivel))
        #GPIO20button()
    if message.topic=="hidroponia/nodo1/aut_on":
        print("Mensaje recibido=", str(message.payload.decode("utf-8")))
        print(GPIO21_state);
        print(GPIO20_State);
        automatico();

        # hidroponia/nodo1/sensores/tds('Mensaje recibido=', '{"tdsagua":29}')
        ph=message.payload.decode("utf-8")
        GPIO21_state=False;

        print(GPIO21_state);
        print(GPIO20_State);
        #y = json.loads(mensaje)
        #ph= y["ph"]
        ph_label.config(text=" PH: " +str(ph))
    if message.topic=="hidroponia/nodo1/aut_off":
        print("Mensaje recibido=", str(message.payload.decode("utf-8")))
        # hidroponia/nodo1/sensores/tds('Mensaje recibido=', '{"tdsagua":29}')
        ph=message.payload.decode("utf-8")
        automatico_state=False
        client.publish("hidroponia/nodo1/disparadores/automat","Apagado")

        #y = json.loads(mensaje)
        #ph= y["ph"]
        ph_label.config(text=" PH: " +str(ph))
```

```
client = mqtt.Client("web_client_py")
client.on_message = on_message
client.connect(broker_address)
client.loop_start() # Inicio del bucle

for topic in mqtt_topics:
    client.subscribe(topic) # Once the client has connected to the broker, subscribe to the topic


ONbutton = tk.Button(master, text="AUT. NIVEL", bg="blue", command=GPIO21button)
ONbutton.grid(row=0, column=0)

OFFbutton = tk.Button(master, text="NIVEL",bg="blue" , command=GPIO20button)
OFFbutton.grid(row=1, column=0)

Exitbutton = tk.Button(master, text="Exit",bg="red", command=master.destroy)
Exitbutton.grid(row=2, column=0)

Testbutton1 = tk.Button(master, text="RELE1",bg="red", command=rele1_button)
Testbutton1.grid(row=3, column=0)

Testbutton2 = tk.Button(master, text="RELE2",bg="red", command=rele2_button)
Testbutton2.grid(row=4, column=0)

Testbutton3 = tk.Button(master, text="RELE3",bg="red", command=rele3_button)
Testbutton3.grid(row=5, column=0)

Testbutton4 = tk.Button(master, text="RELE4",bg="red", command=rele4_button)
Testbutton4.grid(row=6, column=0)

Testbutton5 = tk.Button(master, text="RELES",bg="red", command=rele5_button)
```

```
Testbutton5.grid(row=7, column=0)

Testbutton6 = tk.Button(master, text="RELE6",bg="red", command=rele6_button)
Testbutton6.grid(row=8, column=0)

Motorbutton1 = tk.Button(master, text="MOTOR1",bg="red", command=motor1_button)
Motorbutton1.grid(row=9, column=0)

Motorbutton2 = tk.Button(master, text="MOTOR2",bg="red", command=motor2_button)
Motorbutton2.grid(row=10, column=0)

Motorbutton3 = tk.Button(master, text="MOTOR3",bg="red", command=motor3_button)
Motorbutton3.grid(row=11, column=0)

Motorbutton4 = tk.Button(master, text="MOTOR4",bg="red", command=motor4_button)
Motorbutton4.grid(row=12, column=0)

Automaticobutton = tk.Button(master, text="AUTOMATICO",bg="red", command=automatico)
Automaticobutton.grid(row=13, column=0)

#Fuzzybutton = tk.Button(master, text="Fuzzy",bg="red", command=FuzzyOn)
#Fuzzybutton.grid(row=14, column=0)



temp1_label = tk.Label(master,text =temp1)
temp1_label.place(x = 150,
                  y = 10)
temp2_label = tk.Label(master,text =temp1)
temp2_label.place(x = 150,
                  y = 40)
```

```
temp3_label = tk.Label(master,text =temp1)
temp3_label.place(x = 150,
                  y = 70)
temp4_label = tk.Label(master,text =temp1)
temp4_label.place(x = 150,
                  y = 100)

temp4_label = tk.Label(master,text =temp1)
temp4_label.place(x = 150,
                  y = 100)



                                        #Upper_right.place(relx = 1.0,
#                   rely = 0.0,
#                   anchor ='ne')
tempamb1_label = tk.Label(master,text =tempamb1)
tempamb1_label.place(x = 450,
                  y = 10)

tempamb2_label = tk.Label(master,text =tempamb1)
tempamb2_label.place(x =450,
                  y = 40)

#presamb1_label = tk.Label(master,text =tempamb1)
#presamb1_label.place(x = 450,
   #                y = 70)

#presamb2_label = tk.Label(master,text =tempamb1)
#presamb2_label.place(x = 450,y = 100)

tdsagua_label = tk.Label(master,text =str(tdsagua))
```

```
tdsagua_label.place(x = 450,y = 160)

ph_label= tk.Label(master,text =tempamb1)
ph_label.place(x = 450,y = 190)



nivel_label= tk.Label(master,text =tempamb1)
nivel_label.place(x = 450,y = 220)

PH_SIM_label = tk.Label(master,text ="PH")
PH_SIM_label.place(x = 620,y = 160)
Sal_SIM_label = tk.Label(master,text ="Sal")
Sal_SIM_label.place(x = 620,y = 190)
PH_SIM_label.config(text="PH_SIM: " +str(PH))
Sal_SIM_label.config(text="Sal_SIM: " +str(Sal))



vel1 = tk.Scale(master, from_=-0, to=255)
vel1.set(0)
vel1.place(x = 260, y = 380)
vel2 = tk.Scale(master, from_=0, to=255)
vel2.set(0)
vel2.place(x = 380, y =380)
vel3 = tk.Scale(master, from_=0, to=255)
vel3.set(0)
vel3.place(x = 500, y = 380)
vel4 = tk.Scale(master, from_=0, to=255)
vel4.set(0)
vel4.place(x = 620, y =380)
```

```
PH_T_label = tk.Label(master,text ="PH")
PH_T_label.place(x = 620,y = 280)
Sal_T_label = tk.Label(master,text ="Sal")
Sal_T_label.place(x = 620,y = 310)
#button_ph = tk.Button(master, text="Set PH", command=set_ph)
#button_ph.place(x = 280, y =300)
#button_tds = tk.Button(master, text="Set TDS", command=set_tds)
#button_tds.place(x = 360, y =300)

button_ph = tk.Button(master, text="Set PH", command=set_ph)
button_ph.place(x = 300, y =300)
button_tds = tk.Button(master, text="Set TDS", command=set_tds)
button_tds.place(x = 380, y =300)

button_sim_ph = tk.Button(master, text="SIM PH", command=set_sim_ph)
button_sim_ph.place(x = 130, y =300)
button_sim_tds = tk.Button(master, text="SIM TDS", command=set_sim_tds)
button_sim_tds.place(x = 210, y =300)


button_velocidad1 = tk.Button(master, text="Velocidad 1", command=set_velocidad1)
button_velocidad1.place(x = 260, y =500)
button_velocidad2 = tk.Button(master, text="Velocidad 2", command=set_velocidad2)
button_velocidad2.place(x = 380, y =500)
button_velocidad3 = tk.Button(master, text="Velocidad 3", command=set_velocidad3)
button_velocidad3.place(x = 500, y =500)
button_velocidad4 = tk.Button(master, text="Velocidad 4", command=set_velocidad4)
button_velocidad4.place(x = 620, y =500)
```

```
Hab_motor_button = tk.Button(master, text="HAB. MOTOR",bg="red", command=hab_motor)
Hab_motor_button.grid(row=15, column=0)

Deshab_motorbutton = tk.Button(master, text="DESAB. MOTOR",bg="red", command=deshab_motor)
Deshab_motorbutton.grid(row=16, column=0)

master.mainloop()
```

5.3. Funciones de Membresía – Rasberry Pi

```
import numpy as np
from matplotlib import pyplot as plt
from fuzzy import *


# Variables linguisticas de entrada


# Para el error de PH

# Sensor de PH puede medir de 1 a 14
# Error de PH: -13 a 13
#Paso: 1.7

#ENG = -3.4
#ENP = -1.7
#EC = 0
#EPP = 1.7
#EPG = 3.4
```

```
# Para el error de salinidad

# Sensor de salinidad puede medir de 0 a 1000 ppm
# Error de salinidad: -2000 a 2000 ppm
#Paso: 575

#ENG = -1150
#ENP = -575
#EC = 0
#EPP = 575
#EPG = 1150


# Universo de discurso para el error de salinidad
#Sal_e = np.linspace(-2000, 2000, 100)
Sal_e = np.linspace(-1000, 1000, 50)
# Funciones de membresía para el error de salinidad
#Sal_ENG = trapmf(Sal_e, [-2000, -2000, -1150, -575])
Sal_ENG = trapmf(Sal_e, [-1000, -1000, -750, -250])
#Sal_ENP = trimf(Sal_e, [-1150, -575, 0.0])
Sal_ENP = trimf(Sal_e, [-750, -250, 0.0])
#Sal_EC = trimf(Sal_e, [-575, 0.0, 575])
Sal_EC = trimf(Sal_e, [-400, 0.0, 400])
#Sal_EPP = trimf(Sal_e, [0.0, 575, 1150])
Sal_EPP = trimf(Sal_e, [0.0, 250, 750])
#Sal_EPG = trapmf(Sal_e, [575, 1150, 2000, 2000])
Sal_EPG = trapmf(Sal_e, [250, 750, 1000, 1000])

# Variables linguisticas de salida
```

```
# Para el tiempo de suministro de nutrientes

# Tiempo seteado de 0 a 10s
# Error tiempo: -10 a 10 s
#Paso: 3

#ME = -6
#PE = -3
#NE = 0
#PT = 3
#MT = 6


# Universo de discurso para tiempo de suministro de nutrientes
#Nu_t = np.linspace(-10, 10, 100)
Nu_t = np.linspace(-255, 255, 100)
# Funciones de membresía para tiempo de suministro de nutrientes
#Nu_ME = trapmf(Nu_t, [-10, -10, -6, -3])
Nu_ME = trapmf(Nu_t, [-255, -255, -220, -180])
#Nu_PE = trimf(Nu_t, [-6, -3, 0])
Nu_PE = trimf(Nu_t, [-180, -120, -40])
#Nu_NE = trimf(Nu_t, [-3, 0, 3])
Nu_NE = trimf(Nu_t, [-80, -0, 80])
#Nu_PT = trimf(Nu_t, [0, 3, 6])
Nu_PT = trimf(Nu_t, [40, 120, 180])
#Nu_MT = [trapmf, [180, 220, 255, 255]]
Nu_MT = trapmf(Nu_t, [180, 220, 255, 255])

# Para el tiempo de suministro de agua potable
```

```
# Tiempo seteado de 0 a 10s
# Error tiempo: -10 a 10 s
#Paso: 3

#ME = -6
#PE = -3
#NE = 0
#PT = 3
#MT = 6




# Gráficos de funciones de membresía



# Error Salinidad
plt.plot(Sal_e, Sal_ENG, label="ENG")
plt.plot(Sal_e, Sal_ENP, label="ENP")
plt.plot(Sal_e, Sal_EC, label="EC")
plt.plot(Sal_e, Sal_EPP, label="EPP")
plt.plot(Sal_e, Sal_EPG, label="EPG")
plt.legend(loc='best')
plt.xlabel('error de salinidad')
plt.ylabel('$\mu (e)$')
plt.show()


# Tiempo de suministro de nutrientes
```

```
plt.plot(Nu_t, Nu_ME, label="ME")
plt.plot(Nu_t, Nu_PE, label="PE")
plt.plot(Nu_t, Nu_NE, label="NE")
plt.plot(Nu_t, Nu_PT, label="PT")
plt.plot(Nu_t, Nu_MT, label="MT")
plt.legend(loc='best')
plt.xlabel('Tiempo de suministro de nutrientes')
plt.ylabel('$\mu (Nu_t)$')
plt.show()
```

5.4. Programación Arduino Nano

```
#include <ArduinoJson.h>//libreria para serializar valores de Presion
//#include <SoftwareSerial.h>
//Libreria para expansores
#include <Wire.h>    // Required for I2C communication
#include "PCF8574.h" // Required for PCF8574
//Librerias para control Motores
#include "BTS7960.h"

//Librerias para sensores de temperatura
#include <OneWire.h>
#include <DallasTemperature.h>
//librerias BMP280
#include <Adafruit_BMP280.h>

//definicones Temperarura
#define pinSensoresTemp A2
#define TEMPERATURE_PRECISION 9
```

```
//Variables de sensores de temperatura
// Setear comunicacion OneWire
OneWire oneWire(pinSensoresTemp);
DallasTemperature sensors(&oneWire);
//Direccion de sensores de temperatura
//en las etiquetas el 4 y 2 estan cambiados
DeviceAddress senTemp1 = {0x28, 0x78, 0x0D, 0x76, 0xE0, 0x01, 0x3C, 0x74  };
DeviceAddress senTemp2 = {0x28, 0xBB, 0x01, 0x76, 0xE0, 0x01, 0x3C, 0x93  };
DeviceAddress senTemp3 = {0x28, 0x8B, 0x50, 0x76, 0xE0, 0x01, 0x3C, 0x26  };
DeviceAddress senTemp4 = {0x28, 0xAD, 0xDE, 0x75, 0xD0, 0x01, 0x3C, 0x1D  };
//Variables de temperatura
float temp1, temp2, temp3, temp4;

//Se debe alimentar con 5.0V para mejor rendimiento
//el valor de salida depende el voltaje de entrada
//Se debe usar una fuente de 5V
#define pinSenPH A1
float calibration_value = 21.34;
int phval = 0;
unsigned long int avgval;
int buffer_arr[10], temp;

//Variables sensores BMP
Adafruit_BMP280 bmp1; // Direccion 0x76
Adafruit_BMP280 bmp2; // Direccion 0x77
//Sensor de presion1
Adafruit_Sensor *bmp_temp1 = bmp1.getTemperatureSensor();
Adafruit_Sensor *bmp_pressure1 = bmp1.getPressureSensor();
//Sensor de presion2
Adafruit_Sensor *bmp_temp2 = bmp2.getTemperatureSensor();
```

```
Adafruit_Sensor *bmp_pressure2 = bmp2.getPressureSensor();
//eventos de captura de vallores de temperatura y presion
sensors_event_t temp_event1, pressure_event1, temp_event2, pressure_event2;
float tempAmbiente1, tempAmbiente2, presionAmbiente1, presionAmbiente2;

//Pines de Control Motores BTS7960
//Ojo que estos pines deben ser los mismos para tener el control de velocidad
#define EN1      7
#define L_PWM1   11
#define R_PWM1   10

#define EN2      8
#define L_PWM2   9
#define R_PWM2   6

#define EN3      4
#define L_PWM3   5
#define R_PWM3   3

#define TdsSensorPin A0
#define VREF 5.0      // analog reference voltage(Volt) of the ADC
#define SCOUNT  30           // sum of sample point
int analogBuffer[SCOUNT];    // store the analog value in the array, read from ADC
int analogBufferTemp[SCOUNT];
int analogBufferIndex = 0, copyIndex = 0;
float averageVoltage = 0, tdsValue = 0, temperature = 25;


//Instancias de control de Motores
BTS7960 motorController(EN1, L_PWM1, R_PWM1);
BTS7960 motorController2(EN2, L_PWM2, R_PWM2);
```

```
BTS7960 motorController3(EN3, L_PWM3, R_PWM3);

int velMotor1, velMotor2, velMotor3;

//Intancias de las expansore
PCF8574 expander, expander2;



DynamicJsonDocument docSensoresTx(256);
DynamicJsonDocument docRecibido(256);
//Comunicacion con ESP32
//SoftwareSerial mySerial(12, 13); // RX, TX
long lastMsg = 0;

/*void pinChanged(const char* message, bool pinstate) {
  Serial.print(message);
  Serial.println(pinstate ? "HIGH" : "LOW");
  }*/
void setup() {
  Serial.begin(19200);
  //Inicia comunicacion con esp32
  //mySerial.begin(19200);
  Serial.println("comenzando");

  //Las direcciones de los expnasores debene estar con los jumpers adecuados para la direccion
  expander.begin(0x20);
   Serial.println("CONECTADO1");
  expander2.begin(0x24);
  Serial.println("CONECTADO2");
  //Inicialzar los pines del expnasor como salidas
```

```
  for (int i = 0; i < 8; i++) {
    expander.pinMode(i, OUTPUT);
    expander2.pinMode(i, OUTPUT);
  }
   for (int i = 0; i < 8; i++) {
    expander.digitalWrite(i, HIGH);
    expander2.digitalWrite(i, HIGH);
  }

  //Habilitar los motores
  motorController.Enable();
  motorController2.Enable();
  motorController3.Enable();

   //Iniciar lectura sensores temperatura
   iniciarTemperatura();
    inicioSensBMP();

}

void loop() {

 // if (mySerial.available() ) {
     if(Serial.available() ) {
    //deserializeJson(docRecibido, mySerial);
    deserializeJson(docRecibido, Serial);
    //serializeJson(docRecibido, Serial);
   // Serial.println("");
    if (docRecibido["nombre"] == "modulo1") {
      /* { "nombre":"modulo1", "out1": 1, "out2": 1, "out3": 1, "out4": 1, "out5": 1, "out6": 1, "out7": 1, "out8": 1
}*/
      //Serial.println("Recibo modulo1");
```

```
    expander.digitalWrite(0, docRecibido["out1"]);
    expander.digitalWrite(1, docRecibido["out2"]);
    expander.digitalWrite(2, docRecibido["out3"]);
    expander.digitalWrite(3, docRecibido["out4"]);
    expander.digitalWrite(4, docRecibido["out5"]);
    expander.digitalWrite(5, docRecibido["out6"]);
    expander.digitalWrite(6, docRecibido["out7"]);
    expander.digitalWrite(7, docRecibido["out8"]);

  }
  if (docRecibido["nombre"] == "modulo2") {
    /* { "nombre":"modulo2", "out1": 1, "out2": 1, "out3": 1, "out4": 1, "out5": 1, "out6": 1, "out7": 1, "out8": 1
}*/
  //  Serial.println("Recibo modulo2");

    expander2.digitalWrite(0, docRecibido["out1"]);
    expander2.digitalWrite(1, docRecibido["out2"]);
    expander2.digitalWrite(2, docRecibido["out3"]);
    expander2.digitalWrite(3, docRecibido["out4"]);
    expander2.digitalWrite(4, docRecibido["out5"]);
    expander2.digitalWrite(5, docRecibido["out6"]);
    expander2.digitalWrite(6, docRecibido["out7"]);
    expander2.digitalWrite(7, docRecibido["out8"]);

  }
  if (docRecibido["nombre"] == "motores") {
    /*{ "nombre":"motores", "motor1":100, "motor2":200, "motor3":300,
      }*/
   // Serial.println("Recibo motores");
    //Actualizamos la velocidad de los motores.
```

```
    velMotor1=docRecibido["motor1"];
    velMotor2=docRecibido["motor2"];
    velMotor3=docRecibido["motor3"];

  }
}


long now = millis();
//Publicacion cada 5 seg
if (now - lastMsg > 6000) {
  lastMsg = now;
  docSensoresTx["temp1"] = (int)sensors.getTempC(senTemp1);
  docSensoresTx["temp2"] = (int)sensors.getTempC(senTemp2);
  docSensoresTx["temp3"] = (int)sensors.getTempC(senTemp3);
  docSensoresTx["temp4"] = (int)sensors.getTempC(senTemp4);
  lecturasBMP();
  docSensoresTx["tempamb1"] = (int)tempAmbiente1;
  docSensoresTx["tempamb2"] = (int)tempAmbiente2;
  docSensoresTx["presamb1"] = (int)presionAmbiente1;
  docSensoresTx["presamb2"] = (int)presionAmbiente2;

  docSensoresTx["tdsagua"] = random(100,1000);
  //docSensoresTx["tdsagua"] = (int)lecturaTDS();
  docSensoresTx["ph"] = (int)obtenerPH();

  //serializeJson(docSensoresTx, mySerial);
  serializeJson(docSensoresTx, Serial);
  //Serial.println();
// Serial.flush();
}
```

```
//Serial.flush();
}

void iniciarTemperatura() {
  // Iniciamos sensores de temperatura
  sensors.begin();
  // report parasite power requirements
  /*
  Serial.print("Parasite power is: ");

  if (sensors.isParasitePowerMode()) Serial.println("ON");
  else Serial.println("OFF");
*/
  if (!sensors.getAddress(senTemp1, 0)) /*Serial.println("Unable to find address for Device 0")*/;
  if (!sensors.getAddress(senTemp2, 1)) /*Serial.println("Unable to find address for Device 1")*/;
  if (!sensors.getAddress(senTemp3, 2)) /*Serial.println("Unable to find address for Device 2")*/;
  if (!sensors.getAddress(senTemp4, 3)) /*Serial.println("Unable to find address for Device 3")*/;

  // Establecer resolucion de prescision
  sensors.setResolution(senTemp1, TEMPERATURE_PRECISION);
  sensors.setResolution(senTemp2, TEMPERATURE_PRECISION);
  sensors.setResolution(senTemp3, TEMPERATURE_PRECISION);
  sensors.setResolution(senTemp4, TEMPERATURE_PRECISION);

}

void inicioSensBMP() {
  //if (!bmp.begin(BMP280_ADDRESS_ALT, BMP280_CHIPID)) {
  if (!bmp1.begin(0x76)) {
   // Serial.println(F("Chequear sensor BMP 1"));
    while (1) delay(10);
```

```
  }
  if (!bmp2.begin(0x77)) {
  //  Serial.println(F("Chequear sensor BMP 2"));
    while (1) delay(10);
  }

  /* Default settings from datasheet. */
  bmp1.setSampling(Adafruit_BMP280::MODE_NORMAL,     /* Operating Mode. */
                   Adafruit_BMP280::SAMPLING_X2,     /* Temp. oversampling */
                   Adafruit_BMP280::SAMPLING_X16,    /* Pressure oversampling */
                   Adafruit_BMP280::FILTER_X16,      /* Filtering. */
                   Adafruit_BMP280::STANDBY_MS_500); /* Standby time. */
  bmp2.setSampling(Adafruit_BMP280::MODE_NORMAL,     /* Operating Mode. */
                   Adafruit_BMP280::SAMPLING_X2,     /* Temp. oversampling */
                   Adafruit_BMP280::SAMPLING_X16,    /* Pressure oversampling */
                   Adafruit_BMP280::FILTER_X16,      /* Filtering. */
                   Adafruit_BMP280::STANDBY_MS_500); /* Standby time. */

  //bmp_temp1->printSensorDetails();
//  bmp_temp2->printSensorDetails();
}

void lecturasBMP() {
  //captura de evento del sensor BMP1
  bmp_temp1->getEvent(&temp_event1);
  bmp_pressure1->getEvent(&pressure_event1);
  //captura de evento del sensor BMP2
  bmp_temp2->getEvent(&temp_event2);
  bmp_pressure2->getEvent(&pressure_event2);

 // Serial.print(F("Temperature1 = ")); Serial.print(temp_event1.temperature); Serial.println(" *C");
```

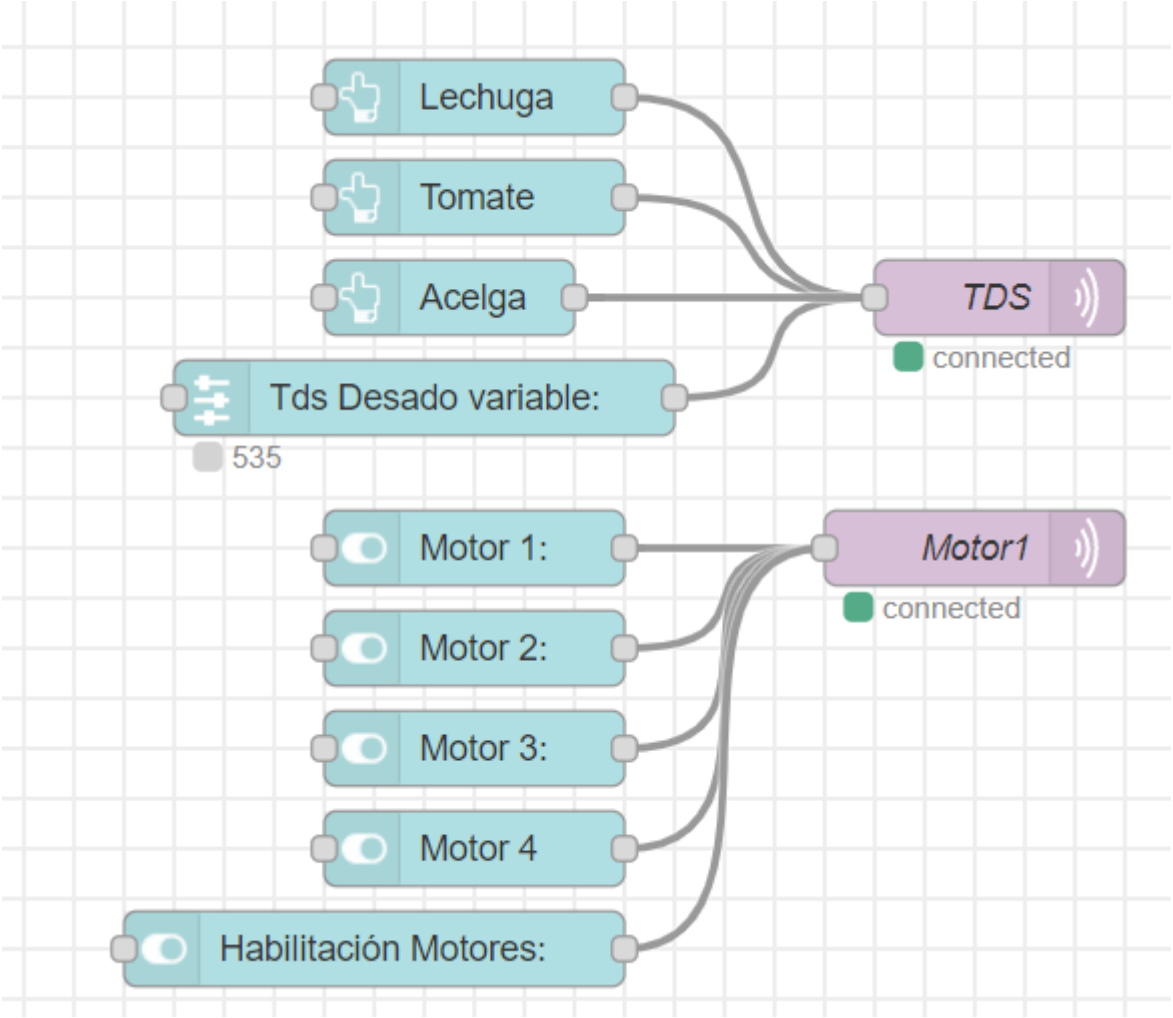
```
 tempAmbiente1 = temp_event1.temperature;
// Serial.print(F("Pressure1 = ")); Serial.print(pressure_event1.pressure); Serial.println(" hPa");
 presionAmbiente1 = pressure_event1.pressure;
// Serial.print(F("Temperature2 = ")); Serial.print(temp_event2.temperature); Serial.println(" *C");
 tempAmbiente2 = temp_event2.temperature;
 //Serial.print(F("Pressure2 = ")); Serial.print(pressure_event2.pressure); Serial.println(" hPa");
 presionAmbiente2 = pressure_event2.pressure;

}

float obtenerPH(){
   for (int i = 0; i < 10; i++)
 {
   buffer_arr[i] = analogRead(pinSenPH);
   delay(30);
 }
 for (int i = 0; i < 9; i++)
 {
   for (int j = i + 1; j < 10; j++)
   {
     if (buffer_arr[i] > buffer_arr[j])
     {
       temp = buffer_arr[i];
       buffer_arr[i] = buffer_arr[j];
       buffer_arr[j] = temp;
     }
   }
 }
 avgval = 0;
 for (int i = 2; i < 8; i++)
   avgval += buffer_arr[i];
```

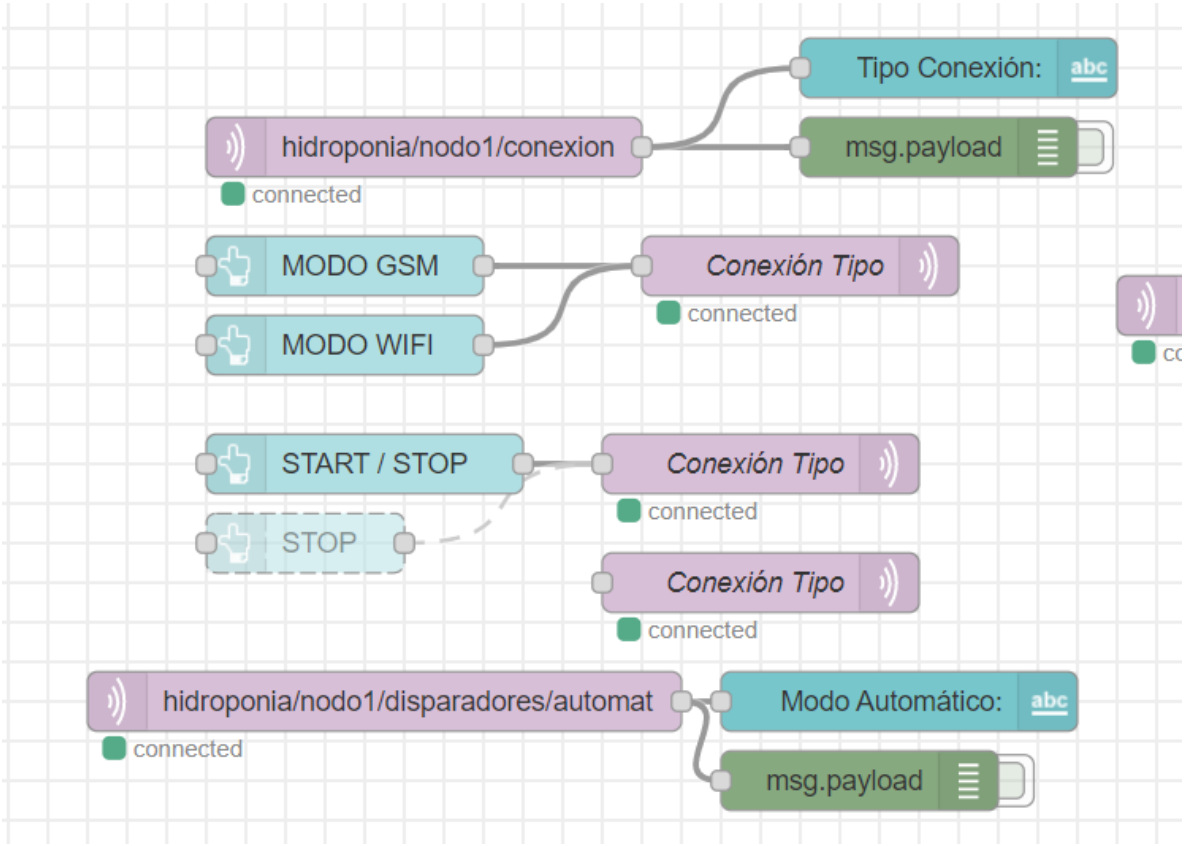
```
 float volt = (float)avgval * 5.0 / 1024 / 6;
 float ph_act = -5.70 * volt + calibration_value;

Serial.print("pH Val:");
Serial.println(ph_act);
 return ph_act;
 //delay(1000);
 }

 int getMedianNum(int bArray[], int iFilterLen)
{
 int bTab[iFilterLen];
 for (byte i = 0; i < iFilterLen; i++)
   bTab[i] = bArray[i];
 int i, j, bTemp;
 for (j = 0; j < iFilterLen - 1; j++)
 {
   for (i = 0; i < iFilterLen - j - 1; i++)
   {
     if (bTab[i] > bTab[i + 1])
     {
       bTemp = bTab[i];
       bTab[i] = bTab[i + 1];
       bTab[i + 1] = bTemp;
     }
   }
 }
 if ((iFilterLen & 1) > 0)
   bTemp = bTab[(iFilterLen - 1) / 2];
 else
   bTemp = (bTab[iFilterLen / 2] + bTab[iFilterLen / 2 - 1]) / 2;
```

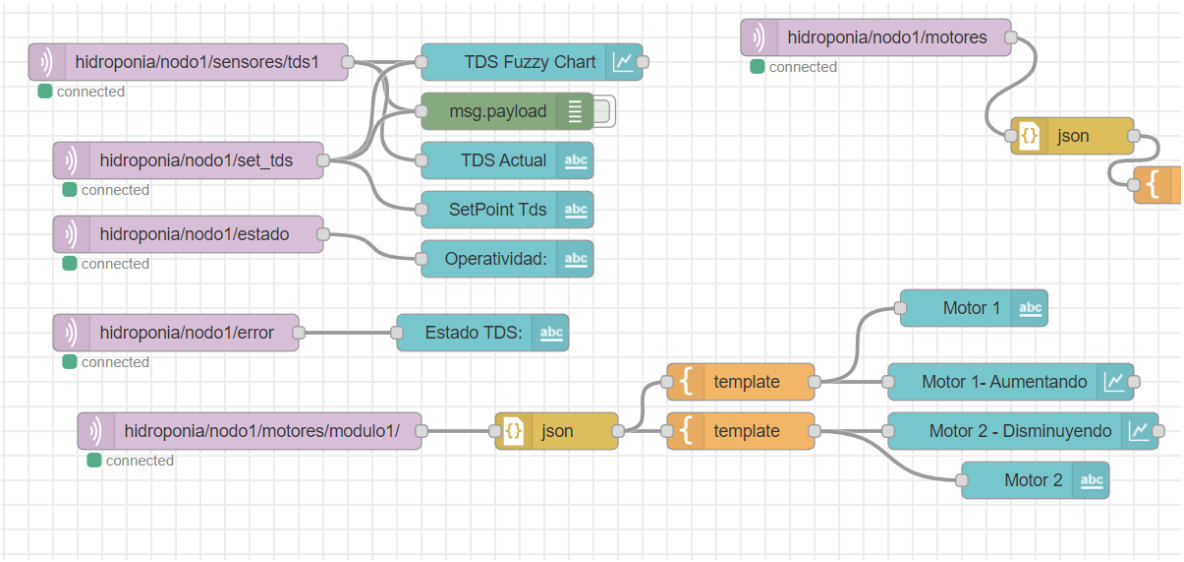
```
  return bTemp;
}


float lecturaTDS() {
  static unsigned long analogSampleTimepoint = millis();
  if (millis() - analogSampleTimepoint > 40U)  //every 40 milliseconds,read the analog value from the ADC
  {
    analogSampleTimepoint = millis();
    analogBuffer[analogBufferIndex] = analogRead(TdsSensorPin);    //read the analog value and store into the buffer
    analogBufferIndex++;
    if (analogBufferIndex == SCOUNT)
      analogBufferIndex = 0;
  }
  static unsigned long printTimepoint = millis();
  if (millis() - printTimepoint > 800U)
  {
    printTimepoint = millis();
    for (copyIndex = 0; copyIndex < SCOUNT; copyIndex++)
      analogBufferTemp[copyIndex] = analogBuffer[copyIndex];
    averageVoltage = getMedianNum(analogBufferTemp, SCOUNT) * (float)VREF / 1024.0; // read the analog value more stable by the median filtering algorithm, and convert to voltage value
    float compensationCoefficient = 1.0 + 0.02 * (temperature - 25.0); //temperature compensation formula: fFinalResult(25^C) = fFinalResult(current)/(1.0+0.02*(fTP-25.0));
    float compensationVolatge = averageVoltage / compensationCoefficient; //temperature compensation
    tdsValue = (133.42 * compensationVolatge * compensationVolatge * compensationVolatge - 255.86 * compensationVolatge * compensationVolatge + 857.39 * compensationVolatge) * 0.5; //convert voltage value to tds value
    //Serial.print("voltage:");
    //Serial.print(averageVoltage,2);
    //Serial.print("V   ");
    Serial.print("TDS Value:");
    Serial.print(tdsValue, 0);
    Serial.println("ppm");
```

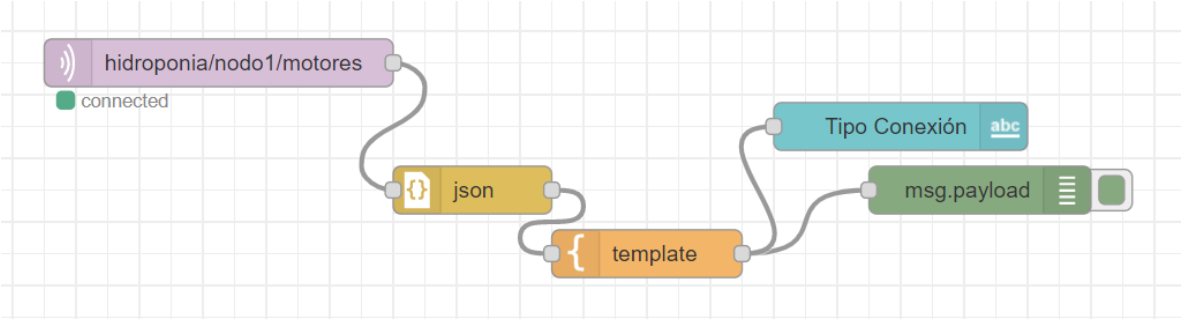
```
  return tdsValue;
 }
}
```

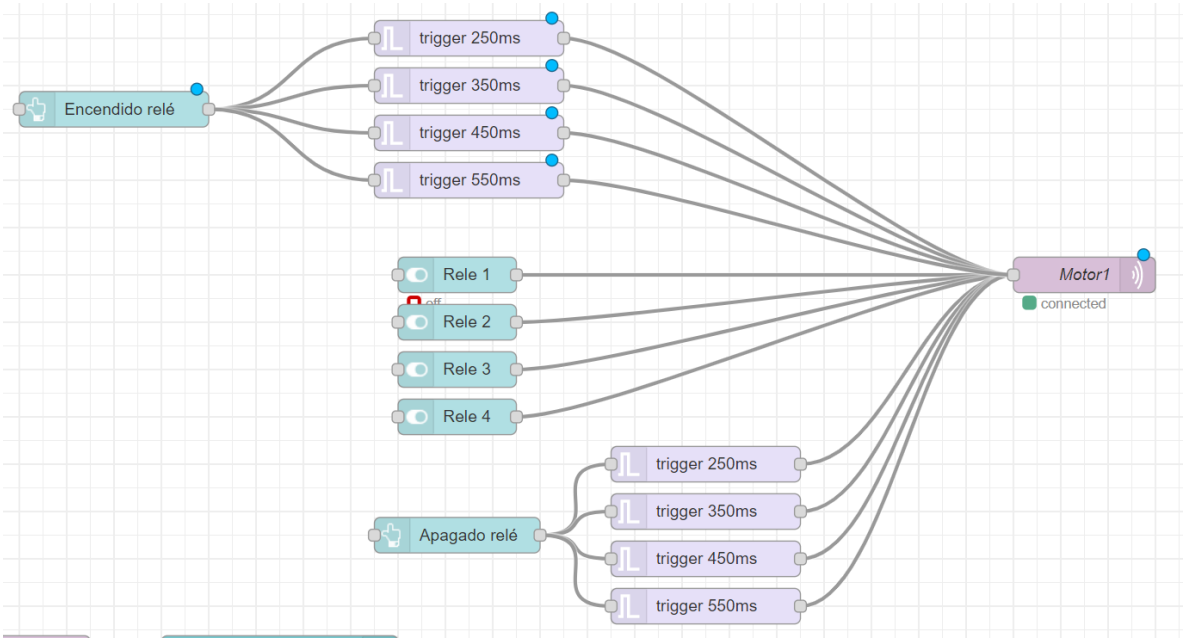
5.5. Diagrama de Flujo en Node-RED




Tipo Conexión:
abc
hidroponia/nodo1/conexion
msg.payload
connected
MODO GSM
Conexión Tipo
MODO WIFI
connected
START / STOP
Conexión Tipo
STOP
connected
Conexión Tipo
connected
hidroponia/nodo1/disparadores/automat
Modo Automático:
abc
connected
msg.payload


hidroponia/nodo1/sensores/tds1
connected
TDS Fuzzy Chart
msg.payload
hidroponia/nodo1/set_tds
connected
TDS Actual
abc
SetPoint Tds
abc
hidroponia/nodo1/estado
connected
Operatividad:
abc
hidroponia/nodo1/motores
connected
json
hidroponia/nodo1/error
connected
Estado TDS:
abc
template
Motor 1
abc
Motor 1- Aumentando
hidroponia/nodo1/motores/modulo1/
connected
json
template
Motor 2 - Disminuyendo
Motor 2
abc


hidroponia/nodo1/motores
connected
json
template
Tipo Conexión
abc
msg.payload


Encendido relé
trigger 250ms
trigger 350ms
trigger 450ms
trigger 550ms
Rele 1
Rele 2
Rele 3
Rele 4
Motor1
connected
Apagado relé
trigger 250ms
trigger 350ms
trigger 450ms
trigger 550ms