PROYECTO PERSONAL
McGarden IOT

FABIÁN ANDRES BEJARANO TOVAR

EDUARD DE LA ROSA
DANIEL MARTÍN PEREZ
ROSER POBLET ESPLUGAS
PROJECTES DE MANTENIMENT ELECTRÒNIC
2WOME 2024-2025
Institut Escola Del Treball

Resumen

Este informe describe el proceso de diseño y desarrollo de un sistema de riego inteligente basado en Arduino y ESP32, elaborado como proyecto final de mantenimiento electrónico.

El objetivo primordial fue crear un sistema de riego controlable tanto remota como manualmente, ofreciendo una interfaz de usuario intuitiva y escalable para futuras expansiones de hardware. Buscando optimizar el uso del agua y facilitar el cuidado de las plantas.

El resultado consiste en una interfaz web que proporciona al usuario información como la humedad del suelo, la temperatura y la humedad ambiental. Esta interfaz también permite la activación manual del riego, controlado mediante un módulo MOSFET y una bomba de agua, cuando la humedad del suelo desciende por debajo del umbral deseado. La arquitectura del sistema se basa en dos componentes principales: una estación de activación, encargada de la medición de datos y la ejecución del riego, y una estación de control, que facilita la gestión de los umbrales de riego y la visualización de información relevante para el usuario.

Resum

Aquest informe descriu el procés de disseny i desenvolupament d'un sistema de reg intel·ligent basat en Arduino i ESP32, elaborat com a projecte final de manteniment electrònic.

L'objectiu primordial va ser crear un sistema de reg controlable tant remotament com manualment, oferint una interfície d'usuari intuïtiva i escalable per a futures expansions de hardware, buscant optimitzar l'ús de l'aigua i facilitar la cura de les plantes.

El resultat consisteix en una interfície web que proporciona a l'usuari informació com la humitat del sòl, la temperatura i la humitat ambiental. Aquesta interfície també permet l'activació manual del reg, controlat mitjançant un mòdul MOSFET i una bomba d'aigua, quan la humitat del sòl davalla per sota del llindar desitjat. L'arquitectura del sistema es basa en dos components principals: d'una estació d'activació, encarregada del mesurament de dades i l'execució del reg, i una estació de control, que facilita la gestió dels llindars de reg i la visualització d'informació rellevant per a l'usuari.

Abstract

This report details the design and development process of an intelligent irrigation system based on Arduino and ESP32, undertaken as the final project for electronic maintenance studies.

The primary objective was to create a remotely and manually controllable irrigation system, offering an intuitive user interface with scalability for future hardware expansions, aiming to optimize water usage and simplify plant care.

The outcome is a web interface providing users with information such as soil moisture, temperature, and ambient humidity. This interface also enables manual irrigation activation, controlled by a MOSFET module and a water pump, when soil moisture falls below the desired threshold. The system architecture comprises two main components: an activation station responsible for data measurement and irrigation execution, and a control station facilitating the management of irrigation thresholds and the visualization of relevant user data.

Indice

Indice de figuras

Indice de Tablas

1. Introducción

La creciente preocupación por la escasez de recursos hídricos y la necesidad de optimizar la producción agrícola han impulsado la búsqueda de soluciones más eficientes y sostenibles en el ámbito del riego, sector en el cual los sistemas tradicionales a menudo resultan en un consumo excesivo de agua y requiriendo una supervisión constante. En este contexto, el Internet de las Cosas (IoT) emerge como una solución prometedora, permitiendo la creación de sistemas inteligentes capaces de monitorizar y controlar diversos parámetros de forma remota y autónoma; Motivación principal que impulsó el desarrollo de este proyecto donde la creciente demanda por controlar y automatizar los aspectos tecnológicos de nuestro día a día, especialmente en entornos cerrados como hogares o pequeñas empresas, donde el control ambiental preciso es cada vez más valorado.

En este sentido, este proyecto se centra en el diseño y desarrollo de un sistema de riego inteligente basado en la plataforma Arduino y el módulo ESP32, con el objetivo principal de optimizar el consumo de agua en pequeños entornos domésticos y facilitar el cuidado de las plantas mediante un control remoto intuitivo. Pudiendo ser ejecutado gracias a los conocimientos adquiridos en los módulos vistos durante el grado, proporcionando la base teórica y práctica necesaria para abordar los desafíos técnicos que se vieron inmersos en la implementación.

Para el diseño de los esquemas eléctricos del sistema, se utilizaron las herramientas de EasyEDA, facilitando la esquematización y la organización de los componentes en las placas de circuito impreso (PCB). Asimismo, el entorno de desarrollo Arduino IDE, junto con sus extensas bibliotecas y las contribuciones de foros y comunidades en línea, resultando fundamentales para resolver cuestiones técnicas específicas y optimizar la implementación del software. Más allá del diseño de un sistema de riego completo que abarque la gestión de humedad del suelo, de los tiempos de riego y la comunicación, este proyecto incorpora una funcionalidad esencial para verificar el estado del sistema incluso en ausencia de una conexión web activa.

2. Objetivos

El presente proyecto se ha concebido con el objetivo de desarrollar un sistema de riego inteligente que supere las limitaciones de los sistemas de riego tradicionales. Estos sistemas a menudo presentan ineficiencias en el uso del agua, requieren una supervisión manual constante y pueden no adaptarse fácilmente a las variaciones en las necesidades de las plantas o las condiciones ambientales. En respuesta a estas deficiencias, se ha decidido plantear una serie de objetivos realistas y alcanzables, enfocados en la creación de un sistema que sea eficaz, adaptable y fácil de usar, proporcionando una solución integral para la gestión del riego.

2.1. Objetivos Generales

- Implementar un sistema de riego inteligente Modular.
- Desarrollar una interfaz de usuario web intuitiva y accesible.
- Optimizar el uso del agua para reducir el impacto ambiental del riego manual.

2.2. Objetivos Específicos

- Diseñar interfaz offline.
- Implementar distintas estaciones de trabajo.
- Disponer de fácil escalabilidad.

3. Marco Teórico

3.1. Antecedentes

La automatización del riego tiene una trayectoria que se remonta a la antigüedad, con sistemas ingeniosos diseñados para optimizar el uso del agua en la agricultura. Civilizaciones antiguas como la egipcia y la romana desarrollaron complejas redes de canales y mecanismos para distribuir el agua a sus cultivos de manera más eficiente que el riego manual. Sin embargo, la integración de tecnologías más avanzadas en el riego doméstico y de jardinería comenzó a tomar forma significativamente más tarde, impulsada por los avances en la electrónica y el control.

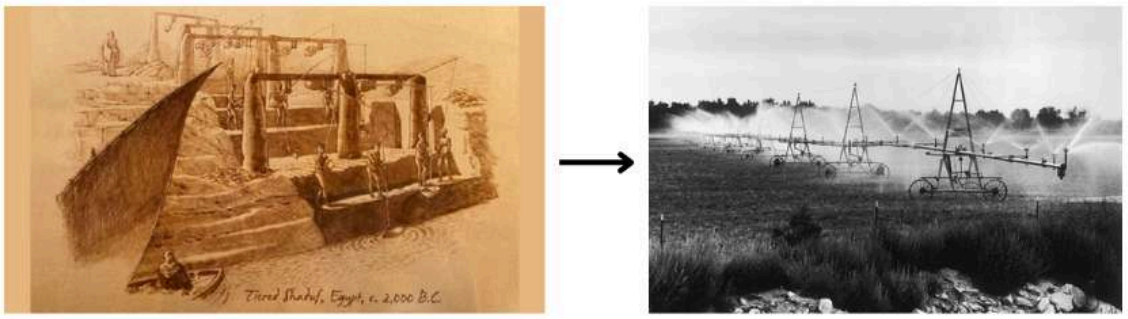


Figura 1: Evolución de sistemas de riego.
Fuente: AuE Riego | Origen del riego.

La evolución de la domótica, cuyo inicio se sitúa a principios de la década de los setenta con la aparición de los primeros dispositivos de automatización en edificios, los sistemas de riego comenzaron a incorporar elementos de control electrónico. Si bien las primeras soluciones eran rudimentarias, marcaban el inicio de una tendencia hacia la automatización para la comodidad y la eficiencia en el cuidado de las áreas verdes. Siendo Save el primer programa de domótica, permitiendo lograr eficiencia y bajo consumo de energía en los sistemas de control de edificios inteligentes. Estas primeras instalaciones empezaron usando el protocolo de comunicación llamado X-10. Este protocolo opera al accionarlo a través de un control remoto. Fue desarrollado en 1976 por Pico Electronics, en Escocia, y aún hoy en día sigue siendo la tecnología más utilizada dentro de la domótica. Utiliza las líneas de bajo consumo para transmitir datos, y en este contexto, los sistemas de riego automático también empezaron a ganar terreno en el ámbito doméstico, aunque inicialmente con funcionalidades limitadas a la programación de horarios.
Pasando por la relación de la domótica con el Internet de las Cosas (IoT) ha marcado un punto de inflexión en el desarrollo de los sistemas de riego inteligente. El IoT, con su capacidad para conectar objetos cotidianos a internet, ha permitido que los sistemas de riego pasen de la simple programación horaria. Ahora, gracias a sensores y la comunicación inalámbrica, es posible monitorizar en tiempo real las condiciones del suelo y el ambiente, ajustando el riego de manera dinámica y

remota. Esta evolución se ha visto impulsada por la miniaturización de los componentes electrónicos y la reducción de costos de los microcontroladores.

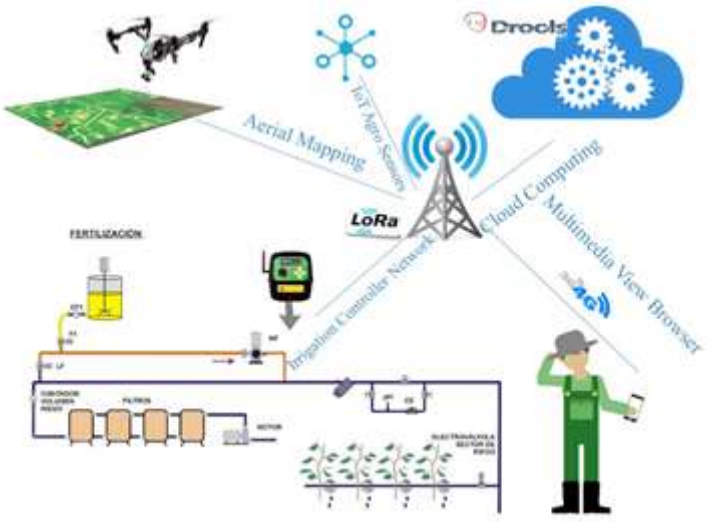


Figura 2: Integración IoT en sistema de riego.
Fuente: Sistema orientado al servicio IoT para el control de la agricultura.

3.2. Definiciones

Internet de las cosas

Por sus siglas en inglés (IoT), es un concepto propuesto por Kevin Ashton en 1999 para referirse a las conexión de objetos cotidianos a través de internet. Su implantación es cada vez mayor y abarca, entre los sectores de la medicina, la industria, el transporte, la energía, la agricultura, las ciudades inteligentes o los hogares.

Figura 3: Internet de las cosas.
Fuente: Internet de las cosas.

Los componentes claves en un sistema IoT son:

- *Dispositivos*: son objetos físicos equipados con sensores, actuadores y la capacidad de conectarse a una red. Los sensores recopilan datos del entorno (temperatura, humedad, movimiento, etc.), mientras que los actuadores

realizan acciones físicas (abrir una válvula, encender una luz, etc.). Estos dispositivos tienen una identidad única en la red, a menudo a través de una dirección IP.

- *Redes de Conectividad:* La comunicación entre los dispositivos y la plataforma IoT se realiza a través de diversos canales, como son; Wi-Fi, Bluetooth, Zigbee, LoRaWAN, NB-IoT, redes celulares (4G, 5G) y Ethernet. La elección de la tecnología de red depende de factores como el alcance, el consumo de energía, el ancho de banda requerido y el costo.
- *Plataformas IoT:* Son la infraestructura en la nube que gestiona y procesa los datos generados por los dispositivos. Estas plataformas proporcionan servicios para la recopilación de datos, almacenamiento, análisis, visualización y gestión de dispositivos. También suelen ofrecer herramientas para el desarrollo de aplicaciones IoT. y
- *Aplicaciones IoT:* Son las interfaces de usuario que permiten a los usuarios interactuar con los datos recopilados y controlar los dispositivos.

Aplicaciones del IoT en la agricultura: Al integrar sensores desplegados en los campos pueden recopilar datos en tiempo real sobre la humedad del suelo, la temperatura ambiente, la radiación solar, la humedad foliar y otros parámetros cruciales para el crecimiento de las plantas, permitiendo obtener información y gestionar de forma óptima el uso del agua, reduciendo el desperdicio y mejorando la salud de los cultivos al evitar el estrés hídrico o el encharcamiento.
También estos sensores y cámaras, pueden utilizarse para detectar la presencia temprana de plagas y enfermedades en los cultivos, activando sistemas de trampas inteligentes que pueden monitorear las poblaciones de insectos, y el análisis de imágenes puede identificar signos de enfermedades en las hojas. Esta detección temprana permite intervenciones más precisas y reduce la necesidad de pesticidas a gran escala.
Utilizando sensores meteorológicos conectados pueden proporcionar datos hiperlocales sobre las condiciones climáticas, lo que ayuda a los agricultores a planificar las siembras, las cosechas y otras actividades agrícolas de manera más eficiente.

Sistemas de riego

Como se ha mencionado anteriormente, los sistemas de riego han evolucionado de distintas formas, de las cuales podrán ser divididas de dos maneras.

Sistemas de riego tradicionales

- *Riego por inundación (o a manta):* Este es uno de los métodos más antiguos y simples. Consiste en liberar grandes cantidades de agua sobre la superficie del suelo, permitiendo que se infiltre por gravedad. A menudo se utilizan canales y surcos para dirigir el flujo del agua a través del campo o la parcela.

- *Riego por Aspersión:* Este método imita la lluvia, distribuyendo el agua a través de aspersores o rociadores que lanzan el agua al aire para que caiga sobre las plantas y el suelo. Existen diferentes tipos de aspersores, desde los estacionarios que cubren un área circular hasta los móviles que se desplazan por el campo.
- *Riego por Goteo (o microirrigación):* Este método suministra agua directamente a la zona de la raíz de las plantas a través de una red de tuberías de bajo diámetro y emisores (goteros) que liberan el agua lentamente, gota a gota. Puede aplicarse en la superficie del suelo o enterrado.

Sistemas de riego inteligente

El riego inteligente en lugar de depender de horarios preestablecidos o de la intervención manual basada en la observación superficial, los sistemas de riego inteligente utilizan datos en tiempo real y algoritmos para determinar las necesidades hídricas exactas de las plantas y ajustar el riego en consecuencia. Su objetivo principal es optimizar el uso del agua, mejorar la salud de las plantas y reducir el desperdicio de recursos. Para adaptar el riego a las condiciones reales, considerando factores como: Humedad del suelo, condiciones meteorológicas, tipo de planta, tipo de suelo, evapotranspiración.

Al analizar estos datos, los sistemas de riego inteligente pueden ajustar la frecuencia y la duración del riego de manera dinámica, asegurando que las plantas reciban la cantidad óptima de agua en el momento adecuado.

Figura 4: Sistema de riego inteligente.
Fuente: Hydrobit | Sistemas de riego inteligentes.

Componentes en un sistema de riego inteligente

Como se ha mencionado en otros apartados, un sistema de riego inteligente puede contar con los siguientes componentes para realizar un óptimo trabajo:

-*Sensores de Humedad del Suelo:* Siendo insertados en el suelo a diferentes profundidades para medir el contenido de agua disponible para las raíces de las

plantas. Proporcionan datos en tiempo real sobre la humedad, lo que permite determinar cuándo y cuánto regar (tratado más adelante).

-*Controladores Inteligentes:* Encargados de recibir los datos de los sensores y las estaciones meteorológicas, ejecutando algoritmos y lógica de control (a menudo configurables por el usuario) para determinar cuándo y durante cuánto tiempo activar el riego. Pueden ser programables a través de interfaces web o aplicaciones móviles y a menudo ofrecen funciones como la programación basada en el clima, el ajuste automático por lluvia y la monitorización del consumo de agua.

-*Actuadores:* Son los dispositivos que realmente controlan el flujo de agua, siendo clasificadas en:

- *Válvulas Solenoides:* Son válvulas electromagnéticas que se abren o cierran eléctricamente bajo el control del programador, permitiendo o interrumpiendo el flujo de agua a diferentes zonas de riego.
- *Bombas de Agua:* En sistemas que requieren presión para la distribución del agua, una bomba controlada por el sistema inteligente se encarga de suministrar el caudal y la presión necesarios.

-*Sistemas de Comunicación:* Permiten la conexión entre los diferentes componentes del sistema y con el usuario, clasificadas como:

- *Cableado:* Para la conexión directa entre sensores, controladores y actuadores, especialmente en sistemas más pequeños o localizados.
- *Inalámbrico:* Permite la comunicación entre dispositivos a distancia y la conexión a internet para el acceso remoto y la integración con plataformas en la nube.

-*Interfaz de Usuario:* Proporciona una forma para que los usuarios configuren el sistema, visualicen los datos recopilados (humedad del suelo, historial de riego, etc.), monitoricen el estado del sistema y realicen ajustes manuales si es necesario.

3.3. Plataformas de Hardware y Software

Hardware

Arduino

Arduino Uno R3

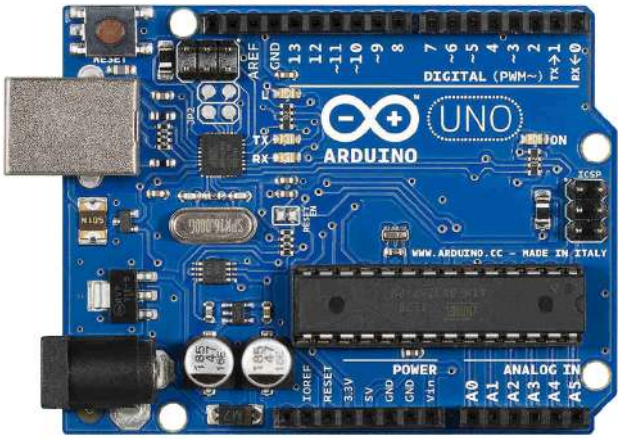


Figura 5: Arduino Uno R3.
Fuente: Arduino Uno R3.

Basado en el microcontrolador ATmega328P que opera a 16 MHz y a 5V, es la placa de desarrollo más icónica y un excelente punto de partida. Ofrece 14 pines digitales de entrada/salida, de los cuales 6 pueden funcionar como salidas PWM, y 6 entradas analógicas con una resolución de 10 bits. Cuenta con 32 KB de memoria Flash para el código, 2 KB de SRAM para datos en tiempo de ejecución y 1 KB de EEPROM para almacenamiento no volátil. Su interfaz USB tipo B facilita la programación y la comunicación serial con el ordenador, haciéndola compatible con una vasta gama de shields y recursos de aprendizaje.

Arduino Mega 2560

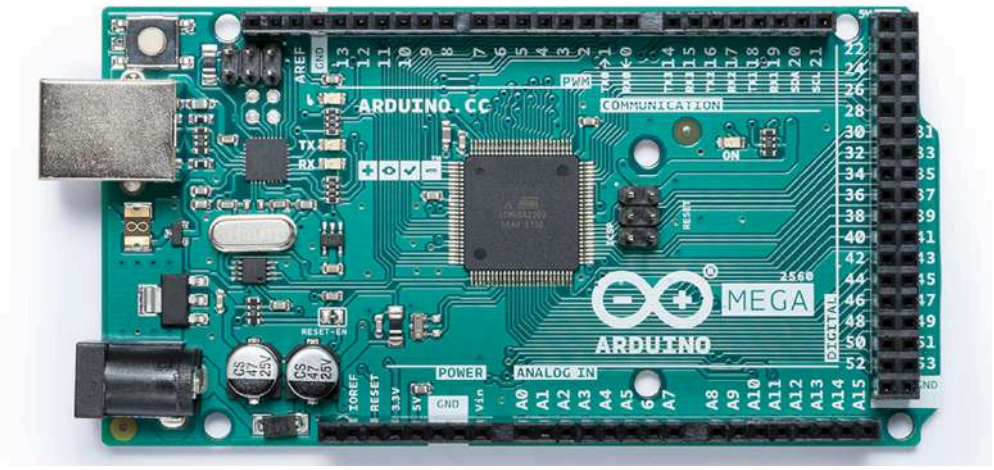


Figura 6: Arduino Uno R3.
Fuente: Arduino MEGA 2560 Rev3.

Es una placa de alto rendimiento impulsada por el microcontrolador ATmega2560 a 16 MHz y 5V, ideal para proyectos complejos que requieren una gran cantidad de conexiones. Proporciona 54 pines digitales de entrada/salida (15 de ellos PWM), 16 entradas analógicas de 10 bits y 4 puertos seriales por hardware (UARTs). Su generosa memoria incluye 256 KB de Flash, 8 KB de SRAM y 4 KB de EEPROM. La interfaz USB tipo B se utiliza para la programación y la comunicación, permitiendo la conexión de numerosos sensores, actuadores y periféricos simultáneamente.

Arduino Nano

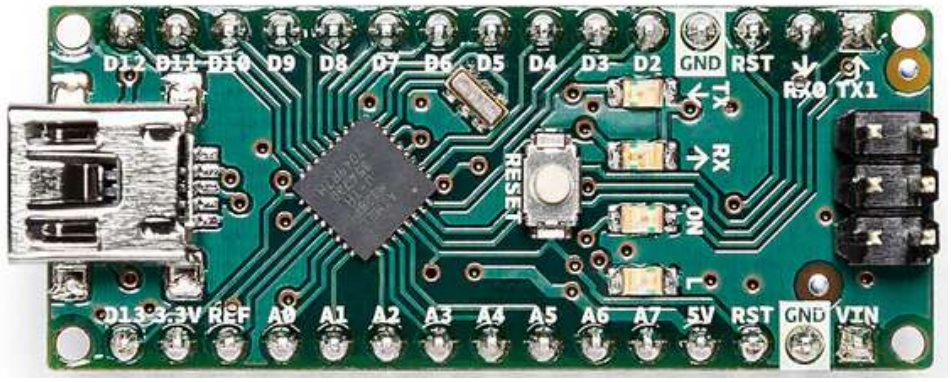


Figura 7: Arduino Nano.
Fuente: Arduino Nano.

Es una versión compacta pero potente, que utiliza el mismo microcontrolador ATmega328P a 16 MHz y 5V que el Uno. A pesar de su pequeño tamaño, ofrece 14 pines digitales E/S (6 PWM) y 8 entradas analógicas de 10 bits. Dispone de 32 KB de Flash, 2 KB de SRAM y 1 KB de EEPROM. Su interfaz USB Mini-B se utiliza para la programación y la alimentación, haciéndola perfecta para proyectos con limitaciones de espacio y sistemas embebidos donde el tamaño es crucial.

Arduino Due

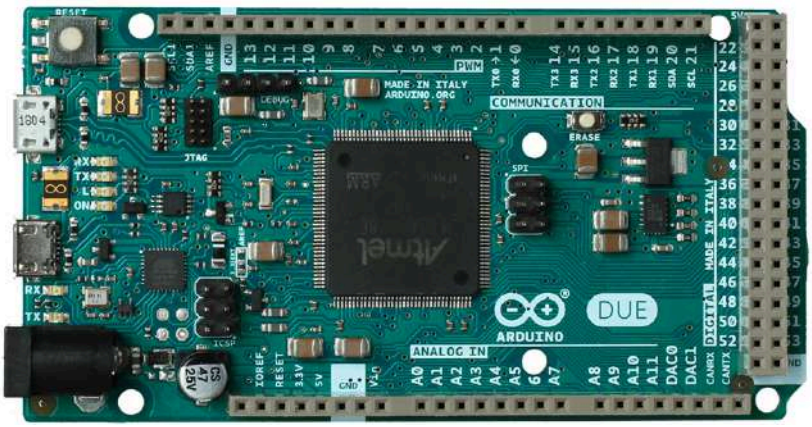


Figura 8: Arduino Due.
Fuente: Arduino Due.

Se distingue por ser la primera placa basada en un microcontrolador ARM Cortex-M3 de 32 bits (AT91SAM3X8E) que opera a una velocidad de reloj de 84

MHz y a 3.3V (con pines no tolerantes a 5V). Ofrece 54 pines digitales E/S (12 PWM), 12 entradas analógicas con una resolución de 12 bits y 2 salidas analógicas (DAC) también de 12 bits. Cuenta con una considerable memoria de 512 KB de Flash y 96 KB de SRAM. Su interfaz USB Micro-B se utiliza para la programación, y también incluye un puerto USB Tipo A que puede funcionar como host.

Arduino Leonardo

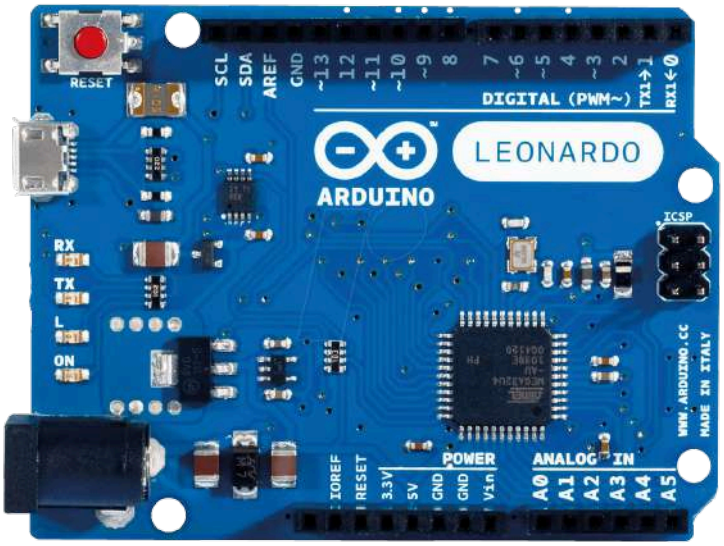


Figura 9: Arduino Leonardo.
Fuente: reichelt elektronik.

ESP

La familia de chips ESP de Espressif se dividen principalmente en dos grandes familias, con varias sub-series y modelos dentro de cada una, que son:

ESP8266

Es un System-on-a-Chip (SoC) de 32 bits, diseñado para aplicaciones de baja potencia conectadas a redes Wi-Fi. Su núcleo es un microprocesador Tensilica L106 RISC que puede operar a frecuencias de reloj de hasta 80 MHz o 160 MHz. Integra una unidad de procesamiento de Wi-Fi (IEEE 802.11 b/g/n de 2.4 GHz) que soporta los modos Station, SoftAP y Station+SoftAP, así como protocolos de seguridad WEP, WPA/WPA2. Típicamente se combina con una memoria flash externa SPI, con tamaños que varían según el módulo (habitualmente desde 512 KB hasta 16 MB), ya que el propio chip tiene una cantidad limitada de SRAM (aproximadamente 96 KB para datos y 32 KB para instrucciones).
Ofreciendo una variedad de interfaces periféricas, que pueden incluir GPIO, UART , SPI , I²C, ADC de 10 bits y PWM . Su consumo de energía es relativamente bajo, con modos de sueño profundo que permiten reducir la corriente a microamperios, lo que lo hace adecuado para aplicaciones alimentadas por batería.
Su placa más popular es la *NodeMCU ESP8266 (v3)*

Figura 10: Placa NodeMCU ESP8266.
Fuente: BricoGeek.com.

Basada en la variante ESP8266-12E, que integra un microprocesador de 32 bits a 80 MHz y conectividad Wi-Fi 802.11 b/g/n. Opera a 3.3V, aunque la placa incluye un regulador para alimentación desde USB Micro-B. Ofrece 17 pines digitales E/S (compartidos con otras funciones) y una entrada analógica de 10 bits, junto con 4MB de memoria Flash para el firmware.

ESP32

Es un SoC de 32 bits de segunda generación, que sucede al ESP8266 y ofrece un rendimiento significativamente mayor y más funcionalidades. Su arquitectura se basa en un microprocesador *Tensilica Xtensa LX6* de doble núcleo, con cada núcleo capaz de operar a frecuencias de reloj de hasta 240 MHz, proporcionando una potencia de procesamiento considerablemente superior. Al igual que el ESP8266, integra conectividad Wi-Fi (IEEE 802.11 b/g/n de 2.4 GHz) con soporte para los mismos modos y protocolos de seguridad. Adicionalmente,incorpora Bluetooth v4.2 BR/EDR y BLE (Bluetooth Low Energy). Cuenta con una mayor cantidad de memoria integrada, típicamente 520 KB de SRAM y 448 KB de ROM. utilizando memoria flash SPI externa, con tamaños que varían según el módulo.
Además incluye un mayor número de pines GPIO con funcionalidades más flexibles, múltiples UARTs, SPIs e I²Cs, hasta 18 canales de ADC de 12 bits, dos canales DAC de 8 bits, PWM con más canales y mayor resolución, interfaces I²S (Inter-IC Sound), Ethernet MAC, CAN 2.0, y sensores táctiles capacitivos.
Debido a sus versátiles características, cuenta con diversos modelos populares, los que destacan:

ESP32-WROOM-32 Dev Kit C

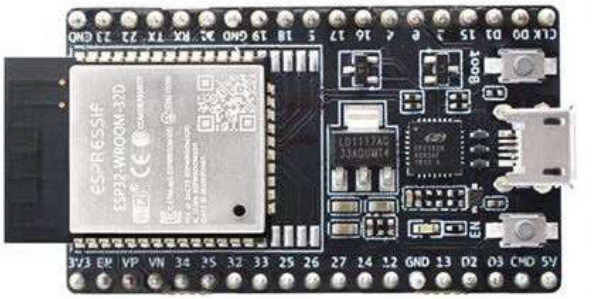


Figura 11: ESP32-WROOM-32 Dev Kit C.
Fuente: NodeMCU (ESP 32).

Es una potente placa de desarrollo que integra la variante del chip ESP32-D0WDQ6 con un procesador de doble núcleo de hasta 240 MHz, 520 KB de SRAM y 4MB de Flash. Ofrece conectividad Wi-Fi 802.11 b/g/n y Bluetooth v4.2 BR/EDR/BLE. La placa proporciona aproximadamente 30 pines digitales E/S, 12 entradas analógicas de 12 bits y 2 salidas analógicas de 8 bits, alimentándose a 3.3V con regulación desde un puerto USB Micro-B. Su versatilidad la hace ideal para una amplia gama de proyectos de IoT con requisitos de mayor procesamiento y conectividad dual.

ESP32-WROVER-E Dev Kit

Figura 12: ESP32-WROVER-E Dev Kit.
Fuente: Olimex Distributor - Mouser Spain.

Amplía las capacidades de memoria con 16 MB de Flash y 8 MB de PSRAM, manteniendo una velocidad de reloj de hasta 240 MHz y 520 KB de SRAM interna. Ofrece las mismas opciones de conectividad Wi-Fi y Bluetooth que el WROOM-32, junto con una cantidad similar de pines digitales y analógicos. La adición de PSRAM la hace adecuada para aplicaciones que requieren más memoria para el manejo de datos complejos, como visión artificial o procesamiento de audio avanzado, con alimentación a 3.3V a través de USB Micro-B.

ESP32-S2 Mini

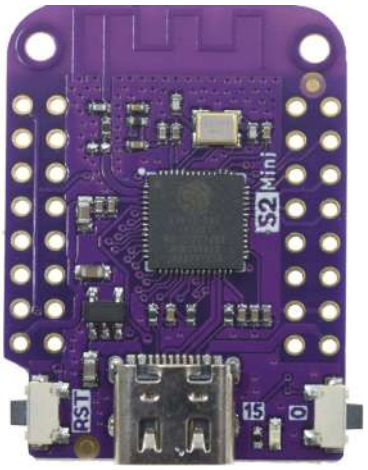


Figura 13: ESP32-S2 Mini.
Fuente: WEMOS documentation.

Es una placa compacta que integra la variante del chip ESP32-S2FN4R2 con un procesador de hasta 240 MHz, 4MB de Flash y 2 MB de PSRAM. Se centra en la conectividad Wi-Fi 802.11 b/g/n (sin Bluetooth Classic) y destaca por su interfaz USB Tipo-C con soporte OTG. Ofrece aproximadamente 27 pines digitales E/S y 18 entradas analógicas de 12 bits, siendo una opción de bajo consumo ideal para dispositivos USB basados en Wi-Fi y aplicaciones de automatización del hogar centradas en Wi-Fi.

Sensores

Los sensores de suelo utilizados en sistemas de riego inteligente pueden clasificarse principalmente por el principio físico que utilizan para medir la humedad. Los más comunes son:

Sensores Resistivos (o Conductivos): Miden la resistencia eléctrica del suelo entre dos electrodos. La resistencia del suelo varía inversamente con su contenido de agua: a mayor humedad, menor resistencia (mayor conductividad). Viéndose afectada por la composición del suelo (especialmente la salinidad y el contenido de nutrientes) y tienden a corroerse con el tiempo debido a la corriente eléctrica que pasa a través del suelo.

Sensores Capacitivos: Estos sensores miden la capacitancia del suelo. El agua tiene una constante dieléctrica mucho mayor que el aire o las partículas del suelo seco. Por lo tanto, a medida que aumenta la humedad del suelo, aumenta la capacitancia medida entre dos placas conductoras del sensor. Siendo más precisos y menos sensibles a la salinidad del suelo. Tienen una menor tendencia a la corrosión al no pasar corriente eléctrica directamente a través del suelo.

Sensores de Tensión (Tensiómetros): Se encargan de medir la tensión matricial o potencial hídrico del suelo, que es la fuerza con la que el agua es retenida por las

partículas del suelo. Consisten en un tubo lleno de agua con una punta porosa que se entierra en el suelo y un vacuómetro en la parte superior. A medida que el suelo se seca, el agua es succionada a través de la punta porosa, creando un vacío que se mide en el vacuómetro (en unidades de centibares o kPa). Proporcionando una indicación directa de la disponibilidad de agua para las raíces de las plantas, independientemente del tipo de suelo. Útiles para determinar cuándo las plantas comenzarán a experimentar estrés hídrico. Su rango de medición es más adecuado para ciertos tipos de suelo (funcionan mejor en suelos arenosos y francos que en arcillosos en rangos de alta tensión).

Sensores TDR (Reflectometría en el Dominio del Tiempo) y FDR (Reflectometría en el Dominio de la Frecuencia): Ambos tipos miden las propiedades dieléctricas del suelo para determinar el contenido volumétrico de agua. Los sensores TDR envían un pulso electromagnético a través del suelo y miden el tiempo que tarda en reflejarse, que está relacionado con la constante dieléctrica del suelo y, por lo tanto, con su contenido de agua. Por su parte los sensores FDR miden el cambio en la frecuencia de resonancia de un circuito causado por las variaciones en la constante dieléctrica del suelo. Son métodos precisos y pueden proporcionar mediciones del contenido volumétrico de agua que son menos sensibles a la salinidad y la temperatura del suelo, siendo a menudo considerados de grado de investigación por su alta precisión.

Actuadores

La activación de los sistemas de riego inteligentes se podrían dividir en dos componentes cruciales, los cuales son:
Actuadores
Encargados de regular y dirigir físicamente el flujo del agua hacia las plantas, los más comunes son:

Válvulas Solenoides: Son válvulas electromecánicas que utilizan un solenoide (una bobina de alambre) para controlar la apertura y el cierre del paso del agua. Cuando se aplica una corriente eléctrica al solenoide, se genera un campo magnético que mueve un émbolo o un diafragma, abriendo o cerrando la válvula. Al interrumpir la corriente, un resorte devuelve la válvula a su estado original. Podrán ser de dos tipos: Normalmente Cerradas (NC) y Normalmente Abiertas (NA).

Bombas de Agua Controlables: Son bombas eléctricas diseñadas para impulsar el agua a través del sistema de riego. En los sistemas de riego inteligente, la activación y desactivación de la bomba está controlada por el controlador. Algunas bombas más avanzadas pueden incluso variar su velocidad de flujo según las necesidades. Existen varios tipos, los cuales son: Bombas Sumergibles, bombas de Superficie, bombas de Presión Constante y bombas de Velocidad Variable.

Válvulas de Zona Motorizadas: Estas válvulas utilizan un pequeño motor eléctrico para abrir y cerrar el paso del agua. El motor se activa mediante una señal eléctrica del controlador y mueve un mecanismo (bola, mariposa, etc.) para controlar el flujo.

Goteros y Microaspersores Controlables: Si bien la mayoría de los goteros y microaspersores son pasivos (liberan agua a un caudal fijo bajo una presión determinada), existen modelos controlables electrónicamente. Estos pueden utilizar microválvulas para regular o interrumpir el flujo de agua a nivel individual de la planta.

Dispositivos de Desvío de Agua (Diverters): Estos dispositivos se utilizan para dirigir el agua hacia diferentes caminos o zonas del sistema de riego. Pueden ser controlados eléctricamente para cambiar la dirección del flujo según las necesidades.

Activadores
Elementos electromecánicos o de estado sólido que permiten o interrumpen el flujo de corriente eléctrica necesario para operar los actuadores antes mencionados, posibilitando así la ejecución precisa de los programas y las decisiones del sistema de control. Estos podrán ser:

Relés Electromecánicos: Un relé es un interruptor electromagnético, donde una pequeña corriente eléctrica que fluye a través de una bobina crea un campo magnético que atrae o libera un contacto, cerrando o abriendo así un circuito de mayor potencia que controla la bomba.

Relés de Estado Sólido (SSR): Utilizan componentes electrónicos de estado sólido (como triacs o MOSFETs) para realizar la conmutación sin partes móviles. Se activan mediante una señal de bajo voltaje y pueden controlar cargas de CA o CC. El controlador envía una señal para activar o desactivar la bomba. Su conmutación rápida puede ser ventajosa en ciertas aplicaciones.

Contactores: Similares a los relés pero están diseñados para conmutar circuitos de mayor potencia, típicamente utilizados para motores eléctricos más grandes como las bombas de agua en sistemas de riego extensos o industriales. Utilizan una bobina electromagnética para accionar contactos que pueden manejar corrientes muy altas. A menudo se utilizan en combinación con un relé de menor potencia que es controlado directamente por el microcontrolador, el cual a su vez activa el contactor.

Software

Protocolos de comunicación

En la actualidad se cuenta con diversos protocolos de comunicación orientada a eventos, entre los más usados en sistemas de riego inteligente se encuentran:

MQTT (Message Queuing Telemetry Transport): Es un protocolo de mensajería ligero basado en el modelo de publicación/suscripción (Pub/Sub). Está diseñado para conexiones con ancho de banda limitado, alta latencia o redes poco confiables. Ideal para la comunicación entre sensores, controladores y plataformas en la nube, especialmente en redes Wi-Fi o celulares con posibles intermitencias. El ESP32 tiene excelente soporte para MQTT a través de diversas bibliotecas.

HTTP/HTTPS (Hypertext Transfer Protocol / HTTP Secure): El protocolo fundamental de la World Wide Web. HTTPS añade una capa de seguridad mediante cifrado SSL/TLS. Se utiliza principalmente para la interfaz de usuario y la integración con servicios en la nube. Una interfaz web o una aplicación móvil pueden usar solicitudes HTTP para obtener datos del sistema de riego (estado actual, historial de riego, lecturas de sensores) desde un servidor (que podría estar ejecutándose en el controlador o en la nube) y enviar comandos de control manual o actualizaciones de la configuración. También añadiendo una capa de seguridad crucial para la comunicación a través de internet.

CoAP (Constrained Application Protocol): Protocolo de transferencia de datos diseñado específicamente para dispositivos con recursos limitados (como microcontroladores) y redes con restricciones (como redes de sensores de baja potencia). Se basa en el modelo REST como HTTP pero es más ligero y utiliza UDP por defecto. Puede ser ventajoso en redes de sensores de baja potencia y recursos limitados, como redes de sensores de suelo alimentados por batería. Su eficiencia y bajo overhead lo hacen adecuado para la comunicación directa entre sensores y un controlador central sin la necesidad de un broker pesado.

LoRaWAN (Long Range Wide Area Network): Un protocolo de capa MAC diseñado para redes inalámbricas de área extensa de baja potencia (LPWAN). Permite la comunicación a larga distancia con un consumo de energía muy bajo. Ideal para aplicaciones de riego inteligente en grandes áreas agrícolas donde la distancia y el bajo consumo de energía son cruciales. Los sensores de suelo, las estaciones meteorológicas o incluso los controladores de riego ubicados en campos extensos pueden comunicarse a través de una red LoRaWAN a una gateway, que luego se conecta a un servidor en la nube. Esto permite la monitorización y el control remoto en áreas donde la infraestructura de Wi-Fi o celular es limitada o costosa**.**

Bluetooth Low Energy (BLE): Una versión de bajo consumo del estándar Bluetooth, diseñada para aplicaciones que requieren un bajo consumo de energía y una comunicación inalámbrica de corto alcance. Se utiliza principalmente para la comunicación local y la configuración de dispositivos en sistemas de riego inteligente. Por ejemplo, una aplicación móvil podría conectarse directamente a un controlador de riego o a un sensor a través de BLE para la configuración inicial, la monitorización en tiempo real dentro del alcance cercano o el control manual. Su bajo consumo de energía lo hace adecuado para dispositivos alimentados por

batería que necesitan comunicarse periódicamente con un dispositivo central cercano.

Zigbee: Es un protocolo de comunicación inalámbrica de bajo consumo y corto alcance, basado en el estándar IEEE 802.15.4, diseñado para redes de malla. En sistemas de riego inteligente, Zigbee puede utilizarse para crear redes locales de dispositivos interconectados, como sensores de suelo, actuadores (válvulas, bombas de baja potencia) y un controlador central.

Tecnologías de interfaz Web

Debido a que el diseño del proyecto hará uso del chip ESP para la transmisión de datos a la web, se describirán únicamente las plataformas que puedan ser usadas con dicho fin.

Frontend (Lado del Cliente)

HTML (HyperText Markup Language): Es el lenguaje de marcado estándar para crear páginas web. Utiliza una serie de etiquetas (tags) para estructurar el contenido, como encabezados, párrafos, imágenes, enlaces, formularios y contenedores. El navegador web interpreta estas etiquetas para renderizar la página que visualiza. La estructura de un documento HTML se basa en un árbol de elementos anidados. Cuando el ESP actúa como servidor web, su firmware genera y envía documentos HTML al navegador del cliente. Estos documentos definen la estructura de la interfaz de usuario que el usuario ve para controlar el sistema de riego (botones de activación manual, visualización de datos de sensores, etc.)

CSS (Cascading Style Sheets): Se utiliza para controlar la presentación visual de los documentos HTML. Permite definir estilos como colores, fuentes, márgenes, rellenos, diseño de la página (usando Flexbox o Grid), y adaptabilidad a diferentes tamaños de pantalla (diseño responsivo).

JavaScript: Es un lenguaje de programación que se ejecuta en el navegador web del usuario. Permite añadir interactividad a las páginas HTML. Se pueden manipular los elementos HTML (DOM), responder a eventos del usuario (clics de botón, cambios en formularios), realizar animaciones y comunicarse con el servidor (el ESP en este caso) de forma asíncrona sin recargar la página (AJAX). Es crucial para crear una interfaz de usuario dinámica para el sistema de riego.

Frameworks y Librerías de JavaScript: Estos proporcionan estructuras de código reutilizables y herramientas para facilitar el desarrollo de interfaces de usuario complejas. React se basa en componentes y un flujo de datos unidireccional. Angular es un framework más completo con una estructura MVC (Modelo-Vista-Controlador). Vue.js es progresivo y fácil de integrar. jQuery simplifica las interacciones con el DOM y las peticiones AJAX. Librerías cómo Chart.js facilitan la creación de gráficos a partir de datos, y Leaflet.js permite integrar mapas

interactivos. Si la interfaz de usuario del sistema de riego necesita ser muy dinámica y compleja (por ejemplo, con múltiples gráficos, controles avanzados, gestión de usuarios), se puede optar por utilizar un framework como React o Vue.js. En este caso, el ESP actúa principalmente como un servidor de API, sirviendo datos en formato JSON cuando la aplicación JavaScript en el navegador lo solicite.

Backend (Lado del Servidor):

Servidor HTTP(S) Integrado: El firmware del ESP8266 o ESP32 incluye bibliotecas que permiten que el chip actúe como un servidor HTTP. Esto significa que puede escuchar las peticiones HTTP que le envían los navegadores web a través de la red (generalmente Wi-Fi) y responder con contenido (archivos HTML, CSS, JavaScript, imágenes, etc.). La adición de soporte HTTPS implica el uso de cifrado SSL/TLS para asegurar la comunicación entre el navegador y el ESP. Pudiendo hospedar una interfaz web completa directamente. Cuando un usuario accede a la dirección IP del ESP en su navegador, el ESP responde enviando el archivo HTML principal. Luego, el navegador solicita los archivos CSS y JavaScript asociados, que el ESP también sirve. Cuando el usuario interactúa con la interfaz (por ejemplo, haciendo clic en un botón), el navegador envía peticiones HTTP al ESP, y el firmware del ESP procesa estas peticiones (quizás activando una bomba o leyendo un sensor) y envía una respuesta (que puede ser otra página HTML o datos en formato JSON).

APIs RESTful: REST (Representational State Transfer) es un estilo de arquitectura para diseñar aplicaciones web. Se basa en recursos (identificados por URLs), métodos HTTP (GET para obtener datos, POST para crear, PUT para actualizar, DELETE para eliminar) y formatos de datos estándar como JSON. Una API RESTful permite que diferentes aplicaciones (en este caso, una interfaz web JavaScript corriendo en un navegador y el firmware del ESP) se comuniquen de manera estandarizada.

WebSockets: Proporciona un canal de comunicación full-duplex persistente entre el cliente (navegador) y el servidor (ESP). Una vez establecida la conexión, ambos lados pueden enviar datos en cualquier momento sin la sobrecarga de encabezados HTTP en cada mensaje. Es muy útil para enviar datos en tiempo real desde el ESP a la interfaz web sin que el navegador tenga que hacer peticiones constantes.

mDNS (Multicast DNS): Permite que los dispositivos en una red local se descubran entre sí utilizando nombres de host amigables (por ejemplo, `mi-esp32.local`) en lugar de direcciones IP numéricas. Funciona sin necesidad de un servidor DNS tradicional en la red local.

4. Desarrollo

4.1. Requisitos

Un sistema domótico se compone de un sistema de control, sensores y actuadores que logran controlar parámetros cruciales en el sistema que esté designado, por el cual se ha establecido una serie de requisitos en la elaboración final del proyecto, las cuales serán:

Interfaces de usuario
Compuesta por:
-Interfaz Web:

- Una interfaz virtual web con la que el usuario pueda integrar el dispositivo a una red de comunicación fija para su administración.
- Cuente con comunicación bidireccional mediante comandos con la que el usuario pueda administrar los servicios del sistema y recibir información detallada.
- Posee botones intuitivos que determinan umbrales para su activación.
- Establecer gráficas con los últimos valores obtenidos de los sensores.
- Disponer de un diseño que pueda ser visto en distintas plataformas.

-Interfaz Física

- Desarrollar una interfaz física mediante pantalla LCD.
- Distribuir un sistema de botones con el fin de controlar la interfaz gráfica y determinar umbrales.
- Contar con un ajuste manual del umbral.
- Entradas físicas para la actualización del código cuando sea requerido.
- Conocer en todo momento la ejecución de las tareas de riego.

Diseño del envoltorio

- Contar con distintos conectores para la conexión de sus sensores.
- Permitir una fácil conexión de alimentación para los requerimientos del sistema.
- Un diseño no tan robusto para su fácil instalación.
- Posee espacio suficiente para su futura ampliación.

Diseño de Placas de Diseño Impreso (PCB)

- Su tamaño no debe ser excesivamente grande, pudiendo ser elaboradas en una placa de 100 mm x 160 mm.
- Permitir una fácil conexión.
- Ser un diseño modular, donde la mayoría de sus componentes no sean fijos completamente.

- Contar con un diseño simple, donde puedan ser reconocidas las conexiones con facilidad.

Protocolos de comunicación

- Disponer de un sistema de comunicación eficiente.
- Aprovechar todas las características por hardware permitidas.

4.2. Materiales

Tomando en consideración los requisitos técnicos y funcionales que abarca el proyecto, se ha decidido utilizar los siguientes materiales, los cuales contarán con una descripción sobre su funcionamiento, pruebas realizadas y limitaciones técnicas que afrontan ellos.

DHT11

El módulo DHT11 es un sensor digital integrado que permite la medición de temperatura y humedad ambiental con salida de señal calibrada. Gracias a que combina un sensor de humedad capacitivo y un termistor NTC para temperatura, junto a un circuito ASIC que digitaliza las mediciones. Su diseño permite transmitir datos hasta 20 metros en condiciones ideales, aunque con limitaciones de precisión (±2°C, ±5% humedad) y velocidad (1 muestra/segundo). En el anexo 1 se encontrará todas sus características técnicas.

Figura 14. Módulo DHT11.
Fuente: Elaboración Propia.

Funcionamiento

Detección de la humedad

El DHT11 detecta el vapor de agua midiendo los cambios capacitivos entre dos electrodos. El componente sensor de humedad es un sustrato con capacidad de retener la humedad con dos electrodos conectados a su superficie. Cuando el

vapor de agua es absorbido por el sustrato, El polímero modifica su Permitividad relativa causando un incremento en la Incremento en la capacitancia entre los electrodos. El cambio en la capacitancia eléctrica entre ambos electrodos es proporcional a la humedad relativa, según la siguiente formula:

$$C = \epsilon_0 \cdot \epsilon_r \cdot \frac{A}{d}$$

Donde:

- C = Capacitancia resultante
- ϵ0 = Permitividad del vacío
- ϵr = Permitividad relativa del polímero (dependiente de la humedad)
- A = Área efectiva de los electrodos
- d = Distancia entre electrodos

Detección de Temperatura

Mediante un termistor NTC (Negative Temperature Coefficient) cuya resistencia eléctrica disminuye exponencialmente al aumentar la temperatura, convierte la resistencia a temperatura usando una tabla de calibración preprogramada, basada en la fórmula:

$$\frac{1}{T} = A + B \cdot \ln(R) + C \cdot [\ln(R)]^3$$

Donde:
T = La temperatura en Kelvin
R = La resistencia medida
A, B, C = Constantes de calibración

Gracias a esta característica, podremos obtener valores entre 0º y 50º de temperatura con una variación de ±2º, con una limitación de tiempo de entre 10 y 15 segundos en cambios detectados.

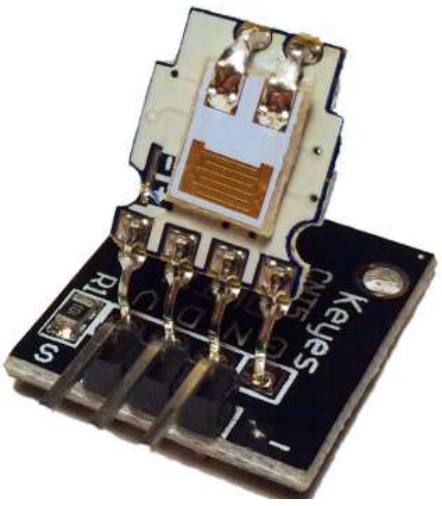


Figura 15: Vista frontal del DHT11.
Fuente: Electrónica Especializada.

Gestión y transmisión de datos

Trabajando con el protocolo de comunicación digital propietario el cual combina hardware dedicado y software embebido, se encarga de calibrar y convertir las señales analógicas a digitales con el fin de enviarlas al Microcontrolador principal (Arduino).

Para la transmisión de los datos digitales, tendrá que recibir una orden de arranque de un pulso bajo de 18 ms seguido de un pulso alto de 20-40µs, para que pase del modo standby (su consumo típico será de 1 mA) al modo funcionamiento el cual podría tomar 1-2 ms dependiendo las condiciones ambientales, cuando esté operativo, lo confirmara con una señal de pulso bajo de 80µs y otra señal de pulso alto de 80µs, esperando a que el Microcontrolador principal complete la señal de arranque, Una vez completada, enviará una señal de respuesta de datos de 40 bits que incluirá la información de los sensores. Una vez que envía los datos, cambiará al modo standby hasta la próxima señal.

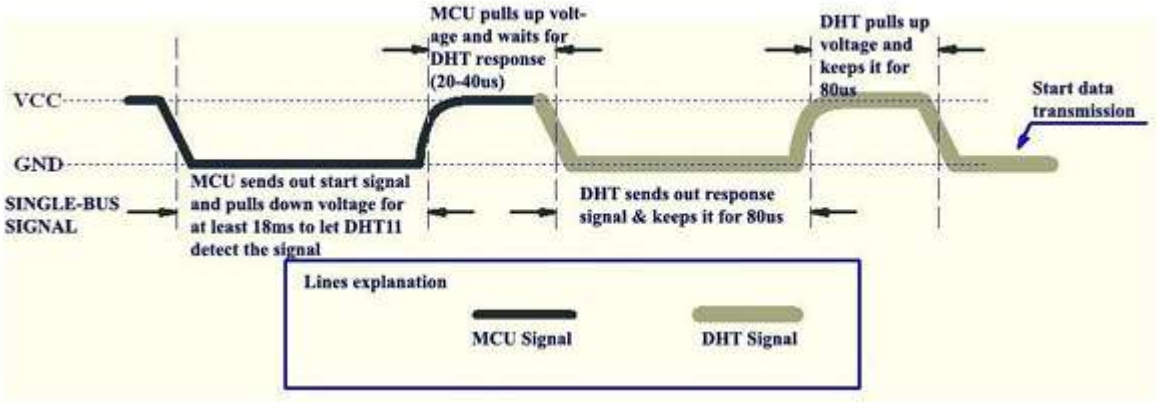


Figura 16: señal de inicio y respuesta del DHT.
Fuente: DHT11 Humidity & Temperature Sensor.

Como se ha mencionado anteriormente, la estructura de transmisión de bits del DHT11 constata de 40 bits, en un tiempo de 4 ms hasta su vuelta a standby. Está compuesta con el siguiente orden:

- 8 Bits para el envío del valor de humedad en formato INT(entero).
- 8 Bits para el envío del valor de humedad en formato decimal (Siempre 00000000 ya que no reporta decimales por su baja resolución).
- 8 Bits para el envío del valor de Temperatura en formato INT(entero).
- 8 Bits para el envío del valor de Temperatura en formato decimal (Siempre 00000000 igual que el anterior).
- 8 Bits para el checksum (Se encarga de calcular los 4 bytes anteriores y que no sean datos erróneos al sumarlo, permitiendo lecturas correctas).
  Ejemplo:

Byte	Contenido	Ejemplo (25°C, 50% H)
1	Humedad entera	00110010 (50%)
2	Humedad decimal*	00000000 (siempre 0)
3	Temperatura entera	00011001 (25°C)
4	Temperatura decimal*	00000000 (siempre 0)
5	Checksum (suma B1+B2+B3+B4)	01001011 (50+0+25+0=75)

Figura 17: Datos enviados del DHT11.
Fuente: Elaboración propia.

Pruebas

Para la verificación del sensor, se utilizó el código “DHT_Verificacion”, agregado en el anexo 2, con la siguiente conexión:

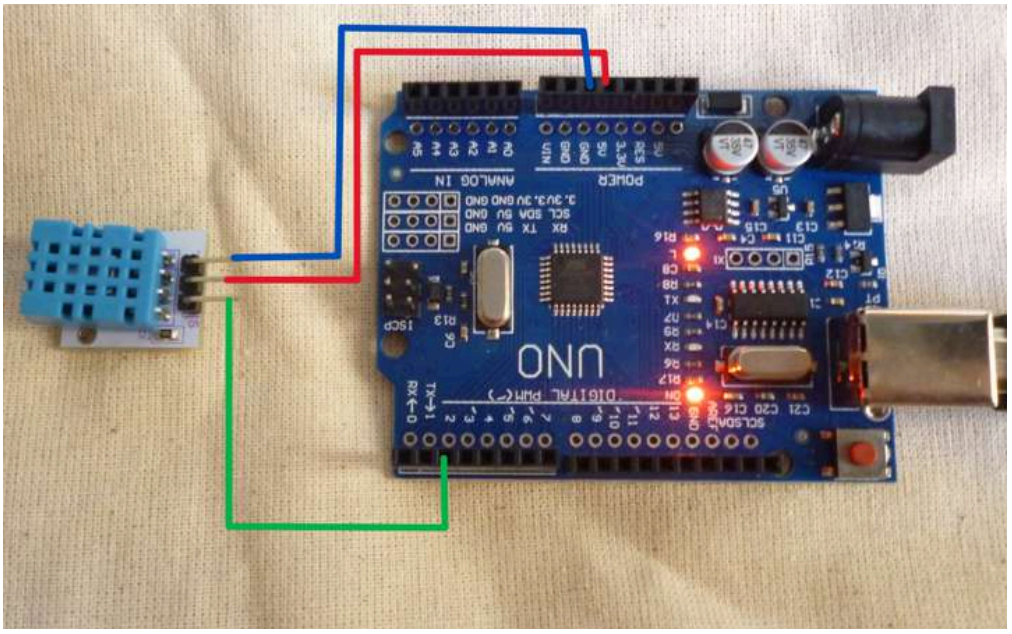


Figura 18: Conexión entre Arduino y DHT11.
Fuente: Elaboración propia.

Suministrando los siguientes datos:

```
Lectura del sensor DHT11
Humedad: 26.90% Temperatura: 30.60°C
Humedad: 26.50% Temperatura: 30.90°C
Humedad: 26.50% Temperatura: 30.90°C
Humedad: 26.40% Temperatura: 30.90°C
Humedad: 26.30% Temperatura: 31.00°C
Humedad: 26.20% Temperatura: 31.10°C
Humedad: 25.90% Temperatura: 31.10°C
Humedad: 31.20% Temperatura: 31.00°C
```

Figura 19: Resultados prueba DHT11.
Fuente: Elaboración propia.

Limitaciones técnicas y fallas comunes

- Cuenta con una precisión y resolución de muestreo bajas, las cuales pueden ser de hasta ±5% para el valor de humedad y de ±2ºC para el valor de

temperatura, además de solo poder entregar valores enteros y no decimales (ej: 22ºC y no 22.5ºC).

- Cuenta con un rango de funcionamiento limitado que lo haría inoperativo en climas bajo cero, ya que su temperatura de trabajo es de 0ºC a 50ºC, como tampoco trabajar en ambientes extremadamente secos o húmedos debido a que su funcionamiento está entre 20% y 80%.
- Puede realizar máximo 1 lectura por segundo y requiere de mínimo 2 segundos entre estas para evitar errores.
- Al contar con el protocolo propietario de 1 cable, se vuelve sensible a interferencias eléctricas en cables largos.
- Requiere un periodo de estabilización de 1 o 2 minutos para ajustar los valores iniciales.
- Ya que cuenta con un polímero higroscópico, este podría degradarse en un periodo de 1 a 2 años si se usa continuamente, causando un error de medición mayor a ±5%, como también su utilización en espacios con ambientes húmedos mayores a 70% causa mediciones erróneas mayores a ±10%.
- Detección limitada de errores, debido a que solo verifica la suma de los 4 bytes anteriores, lo que significa que: no detecta errores si dos bits se cancelan (ej: Byte 1 +1 y Byte 3 -1), no identifica corrupción de datos si el error ocurre en el propio checksum, dando una eficacia sólo de 80%.

Lugares de compra

Distribuidor	Precio
Diotronic	7,56$
Mouser	7,51
Farnell	5,42
AliExpress	1
Amazon	2,79

Tabla 1: Comparación precios DHT11 .
Fuente: Elaboración propia.

Sensor de humedad tierra con LM393

La integración de la sonda HL-69 junto con el módulo comparador LM393 brinda la capacidad de medir la humedad del suelo mediante el principio de conductividad eléctrica entre sus electrodos metálicos, convirtiendo los cambios de resistencia del

suelo en valores de señal analogica y señal digital según el umbral fijado por su potenciómetro (LM393). Sus características técnicas se encontrarán en el Anexo 3.

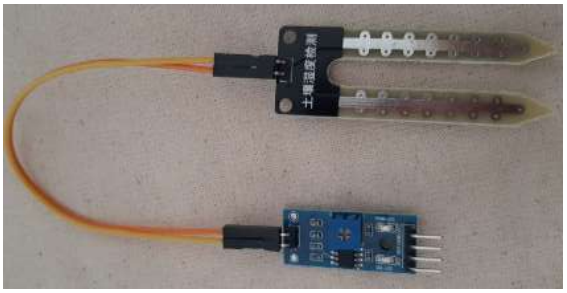


Figura 20: Sonda HL-69 con módulo LM393.
Fuente: Elaboración propia.

Funcionamiento

Sonda HL-69

Permite medir la humedad del suelo basándose en el principio de la conductividad eléctrica, la cual se aplica al utilizar dos electrodos metálicos expuestos que, al insertarse en la tierra, completan un circuito eléctrico.

Debido a que el agua presente en la tierra actúa como electrolito natural, que al contar con mayor humedad en el suelo, los iones del agua facilitan el flujo de corriente, resultando en menor resistencia eléctrica. Por su parte, en suelos secos que cuentan con menor humedad, se aumenta la resistencia eléctrica, reduciendo la conductividad entre los electrodos.

Gracias a este principio podríamos calcular la resistencia del suelo de tal manera:

$$R = \rho \cdot \frac{L}{A}$$

Donde:

- R: Resistencia medida.
- ρ: Resistividad del suelo (depende de humedad y composición).
- L: Distancia entre electrodos.
- A: Área de contacto de los electrodos.

Modulo LM393

Como hemos visto anteriormente, la sonda HL-69 trabaja en conjunto con este módulo para el procesamiento de señales analógicas como digitales. Por lo cual, al ser tratadas de igual manera, siguen dos rutas independientes:

Ruta analógica, pasan por un divisor de tensión formado por la sonda y una resistencia fija de 10kΩ, que podrá ser tratada en un pin ADC del microcontrolador para su procesamiento digital, permitiendo un mayor control.

Ruta digital, la misma señal que ha pasado por el divisor de tensión se deriva hacia el comparador LM393, el cual se encarga de comparar por su entrada IN(-) el valor analogico y por medido de la entrada IN(+)Se ajusta mediante el potenciómetro del módulo. La salida binaria (High/Low) suministrada por éste, podrá ser usada en un

pin digital de un microcontrolador o activar un actuador que no requiere programación.

Procesamiento de la señal Analogica

Teniendo presente el principio físico por el cual opera la sonda y su paso por el divisor de tensión, permite calcular el voltaje que recibe el puerto ADC del microprocesador:

$$V_{AO} = 5V \cdot \frac{R_{\text{sonda}}}{R_{\text{fija}} + R_{\text{sonda}}}$$

Para el caso de un suelo húmedo, donde la resistencia de la sonda disminuye por ejemplo 1kΩ, se calculará de tal manera

$$5V \cdot \frac{1k\Omega}{10k\Omega + 1k\Omega} \approx 0.45V$$

Dando como resultado un voltaje bajo de 0.45V.

En comparación, donde el suelo es seco, la resistencia de la sonda aumenta por ejemplo a 50kΩ

$$5V \cdot \frac{50k\Omega}{10k\Omega + 50k\Omega} \approx 4.16V$$

Destacando que su voltaje alto será de 4.16V.

Esta característica aplicada en el puerto ADC de un microcontrolador (Arduino) que cuente con la resolución de 10 bits, significando que cuando el voltaje sea bajo, producto tener un suelo húmedo, contará con un valor ADC bajo, como si el suelo suele estar más seco, su valor ADC incrementará hasta 1023. La fórmula usada para la realización del mapeo es:

$$\text{Valor ADC} = \frac{V_{out}}{5} \cdot 1023$$

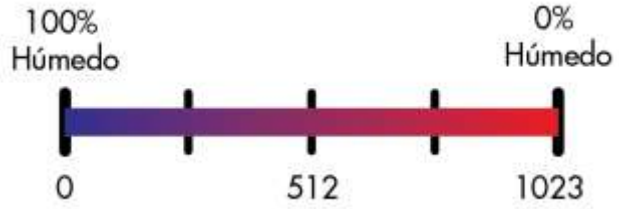


Figura 21: Relación entre la cantidad de humedad y el valor de salida del sensor.
Fuente: Cómo utilizar un sensor de humedad de suelo con Arduino.

Procesamiento de la señal Digital

Utilizando el integrado LM393, el cual se encarga de comparar la señal analógica proporcionada en el divisor de tensión y el punto de disparo ajustado en el potenciómetro para dar señales digitales (High/Low).

El comparador trabaja de la siguiente manera:

- Si la puerta IN+ proveniente de voltaje analógico es mayor a IN- ajustado por el potenciómetro, dará un **HIGH.**
- Si la puerta IN+ es menor que la puerta IN-, proporcionará un **LOW.**

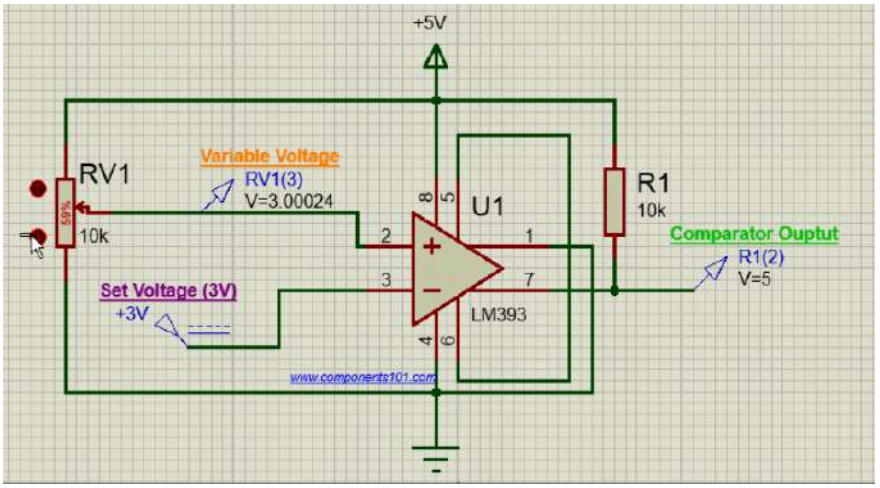


Figura 22: Funcionamiento típico del LM393.
Fuente: Homepage - BharatAgritech.

La comparación de voltajes y su conversión a valores digitales binarios, simplifica su procesamiento y blinda de posibles interferencias en la transferencia de las señales susceptibles a ruido por la longitud de los cables, así como de proporcionar respuestas rápidas de 1.3 µs. Estas características claves, permite la integración en microcontroladores o actuadores, evitando oscilaciones indeseables.

Figura 23: Parte del modulo LM393.
Fuente: Memoria mantenimiento electrónico 2023.

Pruebas

Para conocer el estado óptimo del sensor, se utilizó el código “Sensor_Humedad_Suelo”, agregado en el anexo 4, con la siguiente conexión:

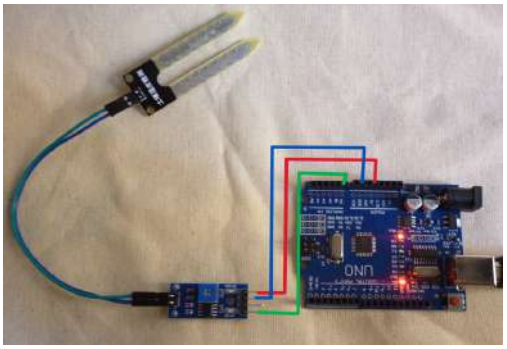


Figura 24: Conexión entre Arduino y módulo LM393.
Fuente: Elaboración propia.

Dando los siguientes resultados en función si está en contacto con suelos completamente húmedos.:

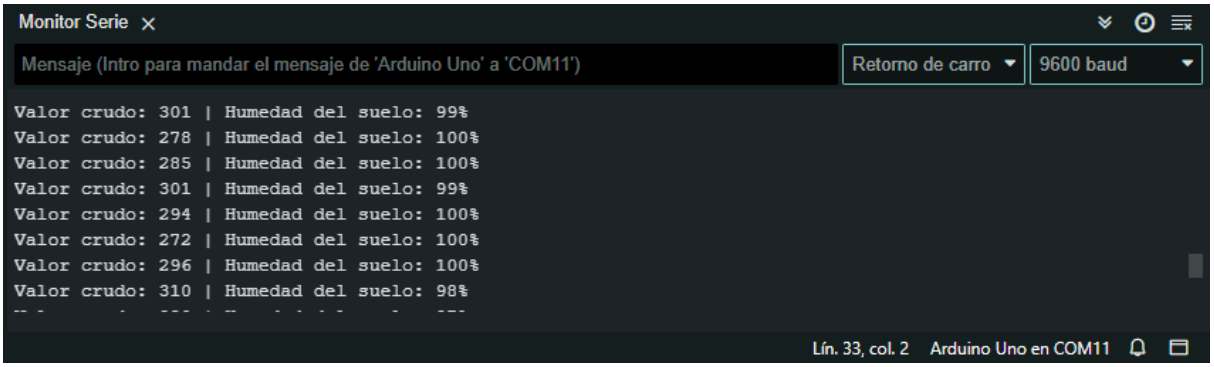


Figura 25: Resultados prueba módulo LM393 en suelos húmedos.
Fuente: Elaboración propia.

También si está completamente seco:

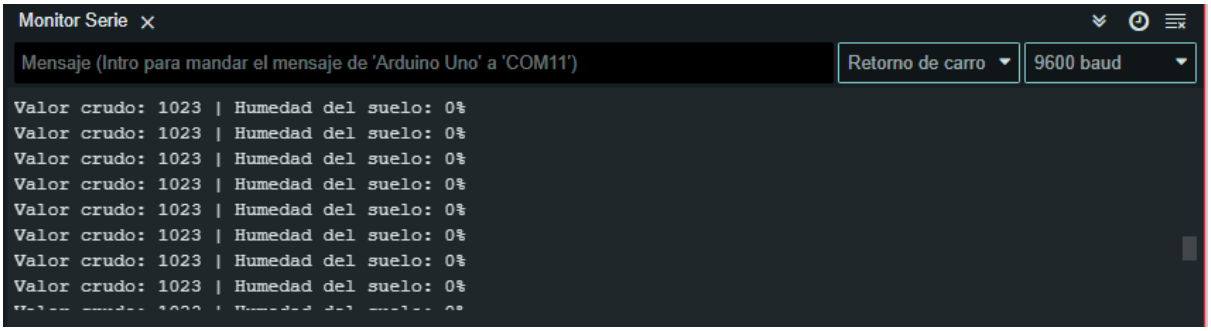


Figura 26: Resultados prueba módulo LM393 en suelos completamente secos.
Fuente: Elaboración propia.

Limitaciones técnicas y fallas comunes

Sonda HL-69

- Al aplicar voltaje CD continuo, los electrodos sufren corrosión acelerada debido a la electrólisis, por lo cual podrá dar mediciones erráticas después de 2-3 meses de uso continuo.

- La resistencia medida podría variar drásticamente según la conductividad del suelo (arcilla o arena), como también de presencia de fertilizantes que alteran la conductividad iónica.
- La curva resistencia-humedad no es lineal, lo que podría ser poco fiable en la toma de valores precisos que oscilan entre 30% y 70%.
- La resistencia podría ver cambiar si la sonda no tiene contacto uniforme con la tierra, debido a bolsas de aire o piedras.
- La resistencia podría variar en función a la temperatura (coeficiente ≈ -0.5%/°C). Causando que a 10ºC pueda tener una resistencia de 20kΩ, pero a 30ºC puedan ser a 18kΩ con el mismo nivel de humedad.
- En suelos muy salinos, los iones crean “puentes conductivos” entre los electrodos, provocando lecturas incorrectas en suelos secos.

Modulo LM393

- Cuando el voltaje de entrada (IN+) está muy cerca del voltaje de referencia (IN-), el comparador puede oscilar entre HIGH y LOW debido a ruido o fluctuaciones mínimas.
- Las entradas IN+ e IN- son sensibles a interferencias electromagnéticas (EMI) o ruido en los cables largos, por lo cual podría causar mediciones falsas o inestables.
- La salida en HIGH/LOW solo puede suministrar hasta 20 mA, debe ir siempre aislada por transistores u optoacopladores para evitar la intercepción de cargas inductivas no deseadas.
- El voltaje de referencia (IN-) indicativo por el potenciómetro podría variar con la temperatura, por ejemplo en ambientes fríos, su valor disminuye, en cambio en calientes, podría aumentar.
- La calibración de los potenciómetros podrían contar con una baja precisión y pueden desgastarse con el tiempo, por el cual el voltaje de referencia puede ser errónea a la deseada.

Lugares de compra

Distribuidor	**Precio**
Diotronic	5,45
AliExpress	1,04
Amazon	5,99 (x5)

Tabla 2: Comparación precios sonda HL-69 + Módulo LM393 .
Fuente: Elaboración propia.

Modulo IRF520

El módulo IRF520 es un driver de potencia basado en MOSFET que permite el control de cargas de alto consumo mediante señales de bajo voltaje que pueden ser suministradas por un microcontrolador (Arduino) de 3.3V a 5V. Gracias a que integra un transistor MOSFET IRF520N de canal N, junto a un circuito de protección con diodo de libre circulación, su diseño soporta la conmutación rápida de cargas de hasta 100V y 9.7A, aunque con limitaciones de disipación térmica. Todas sus características técnicas están en el anexo 4.

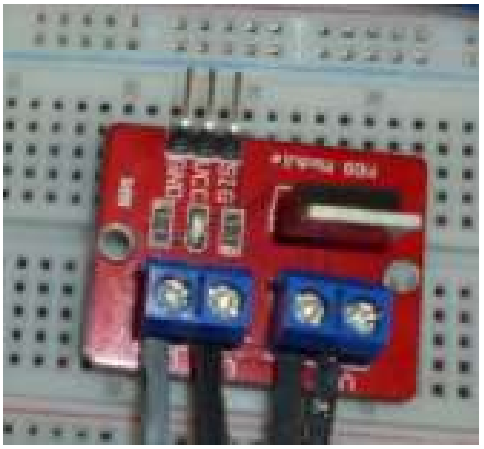


Figura 27: Modulo IRF520.
Fuente: Elaboración propia.

Funcionamiento

Como se ha explicado anteriormente, el Módulo trabaja con el MOSFET de canal N, que actúa como interruptor eléctrico controlado por voltaje, mediante un campo eléctrico generado en el terminal Gate (G), el cual regula el flujo de corriente entre Source (S) y Drain (D).
Gracias a su estructura de Canal N formada principalmente por:

- Metal (Gate): Electrodo de control (aislado del semiconductor por óxido).
- Óxido (SiO_2): Aislante que evita corriente directa hacia el Gate.
- Semiconductor (Silicio tipo P): Substrato donde se forma el canal.
- Regiones tipo N: Source (S) y Drain (D), ricas en electrones.

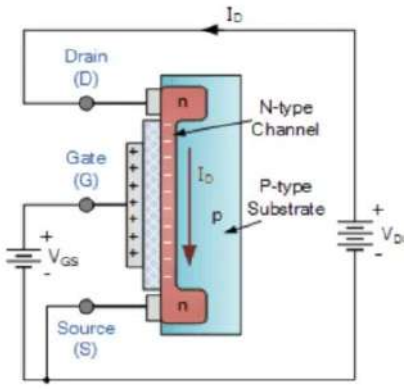


Figura 28: Materiales del Mosfet tipo N.
Fuente: IRF520: todo sobre este transistor MOSFET.

Permite realizar su trabajo, cuando recibe un voltaje positivo en el Gate(Vgs) respecto al Source, creando un campo eléctrico perpendicular al substrato (Silicio tipo P), permitiendo atraer electrones de cargas negativas hacia la superficie del Silicio tipo P. Estos electrones forman un canal conductor tipo N entre el Source y Drain, dando como resultado, el flujo de electrones (Activación del mecanismo).
Pero para que cumpla estas condiciones, el canal de conducción entre el Source y Drain deben cumplir con las características de:

- Si Vgs es mayor del Voltaje umbral (≈ 2-4V), el campo eléctrico será suficiente fuerte para crear este canal, contando con una resistencia entre Drain-Source mínima(~0.27Ω).
- Si Vgs es igual a 0V, no provocará un campo eléctrico, consiguiendo que este canal se bloqueara y delimitando el flujo de corriente.

Para conocer la corriente permitida que circulara por el Mosfet, se usará la siguiente fórmula:

$$I_d = \mu_n C_{ox} \frac{W}{L} \left((V_{gs} - V_{th}) V_{ds} - \frac{V_{ds}^2}{2} \right)$$

- μ_n : Movilidad de electrones.
- C_{ox} : Capacitancia del óxido.
- W/L: Relación ancho/largo del canal.
- V_{ds} :Drain-Source
- V_{th} :Voltaje Umbral

Además de contar con este transistor, el módulo también contiene:

- Resistor Pull-Down de 10kΩ, conectado entre Gate (G) y Source (S). Garantizando que el MOSFET se apague (Vgs = 0V) cuando no hay señal de control.
- Led indicador que muestra el estado de activación del Mosfet

Pruebas

Para verificar el funcionamiento del módulo MOSFET IRF520, cuyo código se encuentra en el anexo 6, se llevó a cabo una prueba. En esta configuración, al presionar un botón, se activará un LED conectado al circuito del MOSFET, lo que indicaba la correcta activación de su puerta. La alimentación del módulo MOSFET se proporcionó a través de una conexión USB.

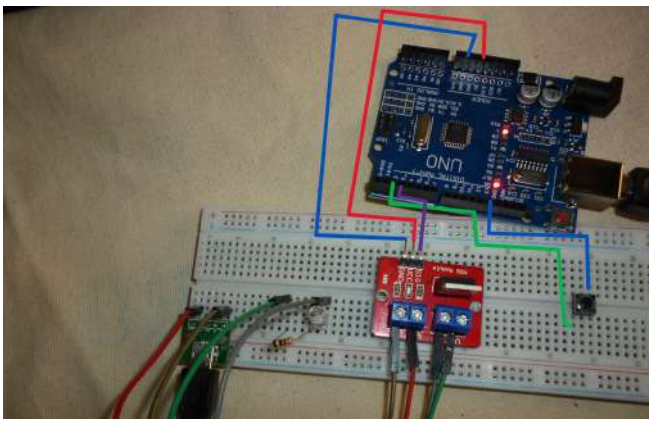


Figura 29: Conexión entre Arduino y módulo IRF520.
Fuente: Elaboración propia.

Se ha realizado un video para realizar esta prueba y estará en el siguiente enlace: https://drive.google.com/file/d/1IBuPxAFSr84I_HdsR7mUYSWEWUPOkEka/view?usp=drive_link

Limitaciones técnicas y fallas comunes

- Debido a que requiere mínimo 4V para la apertura total del Pin Gate, podría no ser suficiente es Señales de 3.3V como lo son del Esp y otros.
- Al no contar con un disipador, podría causar un sobrecalentamiento y daño permanente al suministrar alto amperaje.
- Cuenta con un máximo de 100 kHz en conmutación de frecuencias PWM, por lo cual podría causar una distorsión en señales rápidas.
- El tiempo de conmutación es de 50 ns en ON y de 70 ns en OFF, lo que podría ser un problema en aplicaciones de alta velocidad.
- Ya que el módulo no cuenta con un diodo flyback de protección, podría causar picos de voltaje al desconectar motores o bobinas.

Lugares de compra

Distribuidor	Precio
Diotronic	3,30
tiendotec	2,45
AliExpress	1,21
Amazon	7,39 (x5)

Tabla 3: Comparación precios Módulo IRF520.
Fuente: Elaboración propia.

nRF24L01+

El módulo nRF24L01 es un transceptor de radiofrecuencia de 2.4 GHz que permite la comunicación inalámbrica de datos entre microcontroladores (como Arduino) y otros dispositivos electrónicos. Gracias a que integra el chip nRF24L01 con protocolo propio, junto a un circuito optimizado para modulación GFSK, su diseño soporta transmisiones de hasta 2 Mbps con un alcance práctico de 100 metros (en condiciones ideales). sus características técnicas estarán en el apartado de anexo 7.

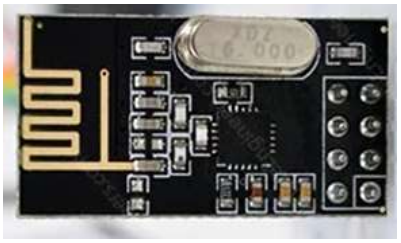


Figura 30: Modulo nRF24L01+.
Fuente: Elaboración propia.

Funcionamiento

Modo Transmisión

Mediante el modo transmisor (TX) envía datos inalámbricos a otro dispositivo mediante paquetes de radiofrecuencia en la banda de 2.4GHz.

Para la realización de este modo, tendrá que ser ajustado ciertos parámetros para su configuración exitosa, mediante comandos SPI desde el microcontrolador (Arduino), los cuales serán:

- Frecuencia de canal, tendrá que ser seleccionado uno de los 126 canales disponibles en el rango de frecuencia de 2.4Ghz (Recomendado el más lejano para evitar interferencias).
- Velocidad de transmisión, estableciendo una de las siguientes velocidades según las necesidades específicas: 250 kbps, 1Mbps y 2Mbps. Aunque partiendo del factor de que a mayor velocidad, menor alcance.
- Potencia de salida, la cual podrá ser ajustada para mejorar el alcance máximo de transmisión del módulo, aunque supondrá un consumo de corriente también. Cuenta con 4 estados, los cuales son:
    - RF24_PA_MIN = -18 dBm (mínima potencia para distancias cortas).
    - RF24_PA_LOW = -12 dBm.
    - RF24_PA_HIGH = -6 dBm.
    - RF24_PA_MAX = 0 dBm (máxima potencia para distancias más largas).

- Dirección del receptor, definiendo una dirección única de 5 bytes en formato hexadecimal.

Es importante que estos datos establecidos en el Microcontrolador de transmisión sean iguales en el dispositivo donde recibirá los datos.

El módulo inicia el proceso de envío de paquetes de datos de 32 bytes, los cuales están estructurados de tal manera:

- Preámbulo, la sincronización del receptor de 1-2 bytes (alternancia de 0s y 1s)
- Dirección: Dirección única del dispositivo receptor conformada por 5 bytes
- Payload: Datos que serán enviados, con un máximo de 32 bytes de tamaño (Si superan este tamaño, podrá ser fragmentados en múltiples paquetes de envío).
- CRC (Cyclic Redundancy Check): Checksum para detección de errores de 1 o 2 bytes.

Estos bits del paquete se convertirán en variaciones suaves de frecuencia (FSK con filtro gaussiano), como pasaran a ser amplificados según el valor anteriormente establecido para ser transmitidos mediante la antena al receptor configurado que se verá más adelante.
Al contar con la espera de ACK (configurada por defecto), el módulo esperará hasta 500 µs una confirmación (ACK) del receptor para confirmar en el sistema que la transmisión fue exitosa, pero después de 3 intentos de comunicación fallida, el módulo notificará su fallo.

Modo recepción
Realizando de forma correcta la transmisión de datos, el modo receptor (RX) capta y decodifica los paquetes de datos suministrados por el transmisor (TX) en el canal dispuesto.
Configurando de forma inicial al igual que en el modo Transmisor, el módulo monitorea su canal establecido en busca de señales válidas, en caso de detectar una señal, iniciará su proceso de demodulación. Este proceso de demodulación usa la técnica GFSK, convirtiendo las variaciones de frecuencia en bits usando el filtro gaussiano, después pasará verificar la integridad de los paquetes recibidos. Si es válido, almacena el payload en su buffer y enviará un ACK al transmisor para confirmar la recepción de sus datos.

Pruebas

Se realizaron pruebas de funcionamiento de este módulo con dos arduinos, siguiendo la configuración detallada en la figura 30. Se incluyó un condensador de 10uF para asegurar la estabilidad de operación.

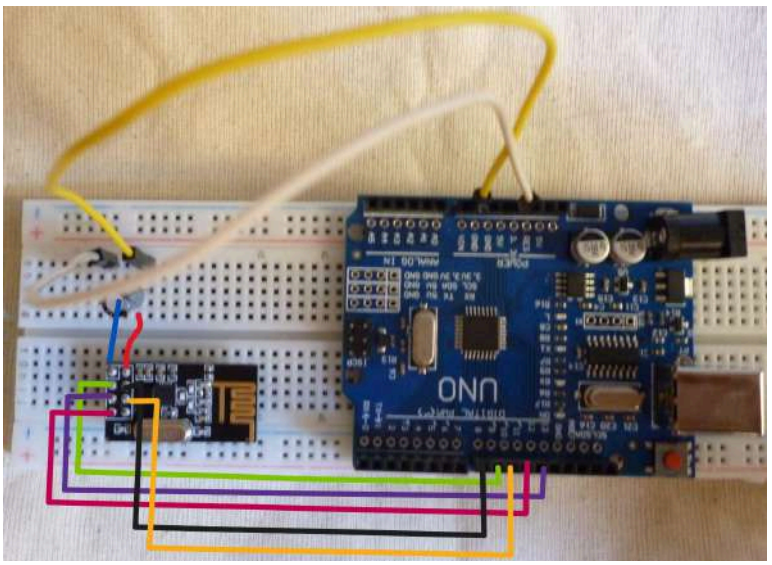


Figura 31: Conexión entre Arduino y módulo nRF24L01+.
Fuente: Elaboración propia.

El codigo de Transmision y recepcion usados en los arduinos se encuentran en el anexo 8, y nos ha arrojado los siguientes resultados:

Mensaje de Transmisión

Figura 32: Resultados de transmisión del módulo nRF24L01+.
Fuente: Elaboración propia.

Mensaje de recepción:

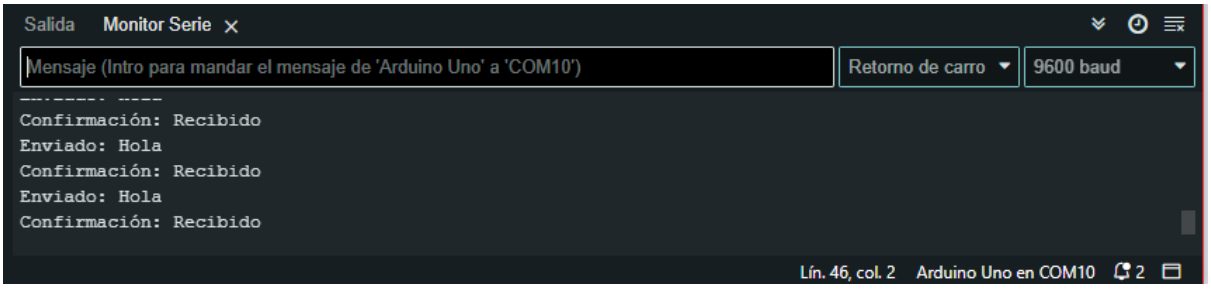


Figura 33: Resultados de la recepción del módulo nRF24L01+.
Fuente: Elaboración propia.

Limitaciones técnicas y fallas comunes

- Pese a contar con un alcance efectivo de hasta 100 metros de distancia, configurados previamente, puede causar pérdida de señal en entornos con obstáculos o interferencias.

- Debido al tamaño del payload de 32 bytes máximos por paquete, puede aumentar la complejidad del código, como una posible pérdida de paquetes si se desea enviar varios.
- Aunque soporta hasta 2 Mbps, esta velocidad máxima reduce el alcance y aumenta la sensibilidad a interferencias.
- Pese a que cuente con hasta 126 canales en el rango de 2.400 a 2.525GHz, podría no ser suficiente en entornos urbanos, donde existan muchos dispositivos que trabajen con este rango de frecuencias al ser de uso libre.
- Solo cuenta con 6 direcciones de recepción, por limitaría su uso en redes multipunto.
- A pesar de contar con una alimentación de 3.3V que puede ser suministrada por un Microcontrolador (Arduino/Esp), necesita capacitores de desacople para no causar picos de tensión.

Lugares de compra

Distribuidor	**Precio**
Diotronic	3,93
Mouser	4,02
AliExpress	2,14
Amazon	7,99 (x5)

Tabla 4: Comparación precios Módulo nRF25L01.
Fuente: Elaboración propia.

LCD16x2 + Modulo I2C

El módulo LCD 16x2 con interfaz I2C es una solución integrada de visualización de datos, basada en el controlador estándar HD44780 y el conversor PCF8574, diseñada para simplificar la conexión con microcontroladores mediante el protocolo I2C. Su arquitectura incorpora 16 caracteres x 2 líneas, soporte para símbolos personalizados (8 bloques definibles por usuario), y comunicación bidireccional a 100 kHz (modo estándar), permitiendo su integración en sistemas embebidos con solo 4 cables (VCC, GND, SDA, SCL). Ideal para proyectos de bajo consumo, su diseño incluye retroiluminación ajustable y direccionamiento configurable (0x20–0x27), asegurando compatibilidad con distintas plataformas. Más sobre sus características técnicas se encuentran en el anexo 9.

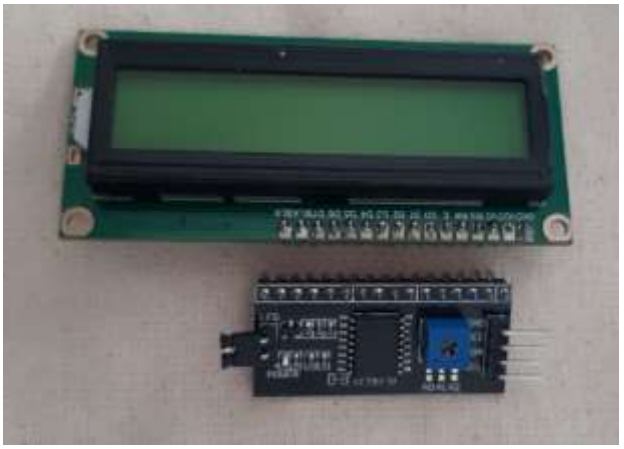


Figura 34: Pantalla LCD16x2 + Modulo I2C.
Fuente: Elaboración propia.

Funcionalidad

Modulo I2C

I2C (Inter-Integrated Circuit) es un protocolo de comunicación serial sincrónico ampliamente utilizado para conectar dispositivos de baja velocidad en sistemas embebidos.
Mediante el protocolo maestro-esclavo, un dispositivo maestro el cual puede ser un arduino controla la comunicación con uno o varios esclavos, utilizando la comunicación half-duplex donde los datos viajan en ambos sentidos, pero no simultáneamente. Este bus está constituido por dos cables (SCL (Serial Clock): Generado por el maestro para sincronizar la comunicación y SDA (Serial Data): Línea bidireccional para transmitir datos).
La estructura de los paquetes de comunicación son las siguientes:

- Condición de START: El maestro baja SDA mientras SCL está alto.
- Dirección del esclavo:
    - 7 bits de dirección (o 10 bits en modo extendido).
    - 1 bit indicando lectura (1) o escritura (0).
- Datos: Paquetes de 8 bits + ACK/NACK por cada byte.
- ACK/NACK: Confirmando la recepción con un bit de acknowledge. Si no responde, el maestro aborta.
- Condición de STOP: El maestro libera SDA mientras SCL está alto.

Cada dispositivo esclavo tiene una dirección única, generalmente en este módulo, la dirección suele ser de 0x20 o 0x27 (siendo verificadas por medio de un código).
En el caso de utilizar este módulo en arduino, será siempre incluida la librería LiquidCrystal_I2C.

Pantalla LCD 16x2

Basado en el controlador HD44780, gestionando la pantalla, interpretando comandos y datos enviados por el microcontrolador. esta característica permite

visualizar caracteres ASCII (letras, números, símbolos básicos) y 8 símbolos personalizables (definibles por el usuario en una matriz de 5×8 píxeles) en un formado de 16 columnas y dos filas (32 caracteres en total).
Conectado mediante los pines Rs (Register Select encargado de Seleccionar entre registro de comandos (RS=0) o datos (RS=1)), RW (Read/Write definiendo si es lectura (1) o escritura (0)), E (Enable para gestionar los pulso alto-bajo necesario para ejecutar comandos) y D4 a D7 (transmisión de 4 bits). Enviando los datos al controlador, encargado de gestionar:

- Memoria DDRAM (Display Data RAM): Almacena los caracteres a mostrar (80 bytes)
- Memoria CGRAM (Character Generator RAM): Para crear caracteres personalizados (64 bytes)
- Registros de control:
    - IR (Instruction Register): Para comandos.
    - DR (Data Register): Para datos.

Pruebas

Usando el arduino uno, se ha realizado la siguiente conexión para la visualización de datos:

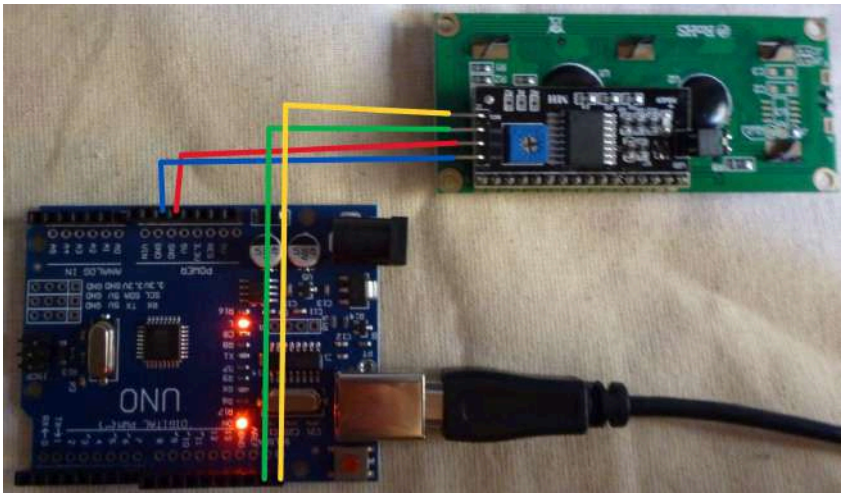


Figura 35: Conexión entre Arduino y Módulo I2C.
Fuente: Elaboración propia.

El código fue elaborado con el fin de mostrar mensajes en la pantalla y verificar la fluidez de ella, por lo cual se añadió la herramienta de contar los segundos desde que fue encendida. El código se encontrará en el anexo 10.

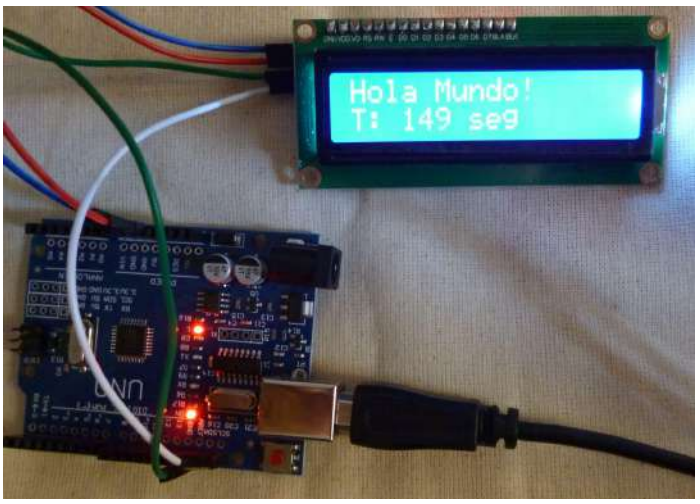


Figura 36: Resultados de comunicacion Arduino y pantalla LCD.
Fuente: Elaboración propia.

Limitaciones técnicas y fallas comunes

- No es ideal para aplicaciones que requieren actualizaciones rápidas (lecturas de sensores de alta frecuencia), ya que el bus I2C es más lento que una conexión directa en paralelo.
- Ya que el protocolo I2C cuenta con hasta 128 direcciones distintas, podría causar conflictos con otros módulos que cuenten con la misma dirección establecida.
- Algunas versiones antiguas de la librería LiquidCrystal_I2C tienen limitaciones para manejar caracteres personalizados (custom chars) o símbolos no ASCII.
- En condiciones de alta luminosidad ambiental (como luz solar directa), la pantalla puede volverse ilegible debido a su bajo contraste.
- Debido al uso de la retroiluminación LED, podría consumir energía adicional (característica esencial en proyectos IOT).
- Debido al ruido eléctrico que puede ser producido por voltajes inestables, podría causar una inicialización incorrecta del LCD, como también mostrar texto distorsionado o caracteres raros.

Lugares de compra

Distribuidor	**Precio**
Diotronic	9,26
AliExpress	1,18
Amazon	6,92

Tabla 5: Comparación precios pantalla LCD16x2 junto a modulo I2C.
Fuente: Elaboración propia.

Bomba de agua sumergible 5v

Es un dispositivo compacto de bombeo de líquidos, diseñado para operar con bajo consumo energético y ser compatible con sistemas electrónicos de control. Basada en un motor de corriente continua, su arquitectura permite un flujo de 80-120 L/h, con una elevación máxima de 40 cm, asegurando un rendimiento eficiente en aplicaciones de riego automatizado, acuarios y dispensadores de líquidos.
Su diseño incorpora una carcasa plástica resistente, con un diámetro de salida de 7.5 mm externo / 5 mm interno, optimizando la conexión con tuberías estándar. Más sobre las características técnicas en el anexo 11.

Funcionalidad

La bomba opera gracias a un motor DC alojado dentro de una carcasa plástica impermeable. Al aplicar un voltaje de 5V, el motor activa un rotor interno que gira rápidamente, creando un efecto de succión en la entrada de agua y generando presión en la salida, siguiendo el siguiente patrón:

- *Entrada de agua*: El líquido ingresa a través de una abertura situada en la parte inferior de la bomba.
- *Movimiento del rotor*: El motor gira un impulsor o hélice, creando una fuerza centrífuga que mueve el agua hacia la salida.
- *Salida del agua*: La presión generada por el rotor expulsa el agua por el conducto de salida.

Limitaciones técnicas y fallas comunes

- Su capacidad de elevación está restringida a 40 cm, por lo que no es ideal para sistemas que requieran presión alta.
- Su flujo máximo de 80-120 L/h es adecuado para aplicaciones pequeñas, pero insuficiente para sistemas de riego grandes.
- Está fabricada con carcasa de plástico, lo que la hace menos resistente en entornos exigentes en comparación con modelos de materiales metálicos.
- Solo funciona sumergida en agua, por lo que si se activa sin estar sumergible, puede estropearse.
- Su funcionamiento óptimo está dentro de un rango de temperatura de -20°C a 50°C, lo que la hace vulnerable a ambientes extremos.
- Al ser una bomba de pequeño tamaño, su motor puede generar un leve zumbido que podría ser molesto en algunas aplicaciones.
- La acumulación de sedimentos o partículas puede bloquear la entrada de agua, reduciendo su rendimiento o utilizando.
- En usos prolongados sin ventilación adecuada, el motor puede calentarse excesivamente y disminuir su eficiencia.

Lugares de compra

Distribuidor	Precio
Diotronic	9,50
AliExpress	1,39
Amazon	8,99

Tabla 6: Comparación precios Bomba de agua sumergible 5V
Fuente: Elaboración propia.

4.3. Diseño

El diseño final fue elaborado con la búsqueda de un equilibrio entre funcionalidad y ampliaciones futuras, por la cual se han modificado componentes que nos permiten conseguir mayores márgenes de operación durante el proceso, junto que simplifiquen la construcción, requiriendo menos componentes externos con las operaciones similares. Reflejado en el siguiente apartado.

Diseño de prototipo

Primera aproximación

En un principio, el objetivo principal ha consistido en dotar de una interfaz de comunicación inalámbrica al sistema. Por ello, el primer prototipo contaba con la placa NodeMCU que incorpora un ESP8266.

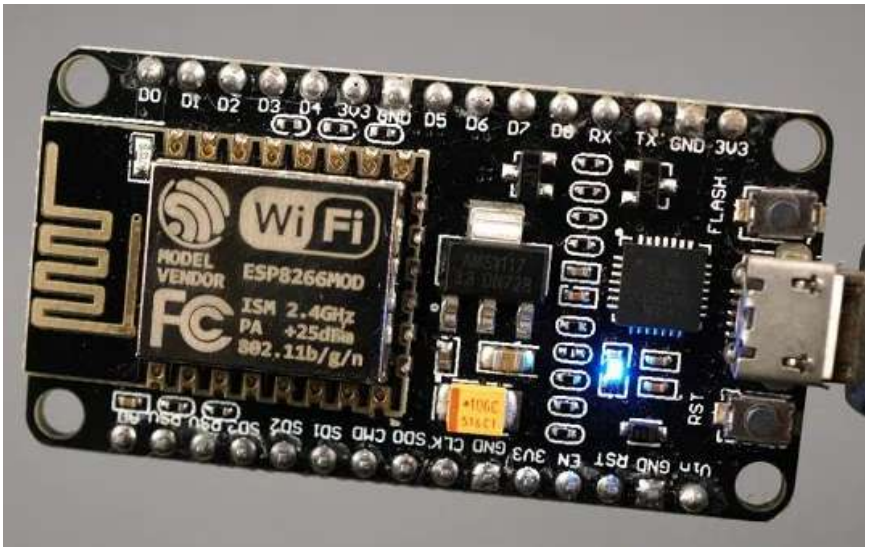


Figura 37: NodeMCU con ESP8266.
Fuente: [NodeMCU ESP8266: Guía completa, ejemplos y experiencia real].

Esta placa NodeMCU proporciona conectividad WiFi, contiene un microchip ESP8266 y viene integrado con una memoria Flash externa, típicamente de 4MB. El NodeMCU no es solo un chip, sino una placa de desarrollo que incluye reguladores

de voltaje, conversor USB-serial y 11 pines GPIO digitales y uno analogico, ideal desde un principio para la utilización de pines con el fin de crear una interfaz física en el lugar donde vaya a estar situado.

Esta placa NodeMCU, fue considerada inicialmente por su integración WiFi y poder diseñar su Software en la plataforma arduino IDE.

Las limitaciones físicas de este módulo empezaban a ser evidentes, al momento de realizar distintas conexiones con botones que deberían emular un interfaz junto con una pantalla LCD, la cual también era un obstáculo para alimentarla, ya que solo suministra 3,3V , junto a que la información para su funcionamiento es suministrada por un módulo I2C y el NodeMCU no cuenta con un protocolo de comunicación para este, el cual tendría que ser puesto por software (aumentando peso en la ejecución del código). Uno de sus principales obstáculos también proceden a la hora de incorporar el módulo nRF24L01 para la transmisión de datos con el otro módulo, el cual se quedaba sin pines necesarios para su conexión.

Segunda aproximación

Debido a las limitaciones mencionadas anteriormente, se decidió usar la placa ESP32 Dev Module.

Figura 38: ESP32 Dev Module.
Fuente: ESP32 Board with EzloPi for Internet of Things (IoT).

La placa de desarrollo ESP32 DEv Module integra conectividad Wi-Fi y Bluetooth, gracias a su microchip ESP32-D0WDQ6 de doble núcleo. Cuenta con 4MB de memoria flash externa, un conversor USB-serie CP2102 para facilitar la programación y comunicación, y 36 pines GPIO. Estos pines GPIO pueden configurarse como hasta 18 canales ADC de 12 bits, 2 DAC y 10 pines touch capacitivos, dependiendo de la configuración. Además, ofrece una amplia variedad de interfaces de comunicación, incluyendo 3 UART hardware, 2 I2C, 4 SPI, 2 I2S (para audio digital) y 1 CAN bus (con controlador integrado).

Gracias a las características que cuenta esta placa, se decidió integrar los Touch capacitivos al proyecto y hacer que el diseño de la interfaz donde se va ubicar, sea táctil y no tenga que presionar botones físicos, permitiendo un diseño más profesional sin necesidad de añadir materiales, también se ha decidido integrar 3 led por la posibilidad de tener más pines disponibles.
Debido a que cuenta con un pin especial de 5V, permite alimentar la pantalla LCD.
Al contar con un microchip más robusto de 2 núcleos, ha permitido que la transmisión de datos mediante WIFI, la gestión del módulo nRF24, como de la interfaz integrada sea más ligera, evitando la saturación incluso con múltiples órdenes.

Funcionamiento

Como se menciona en el apartado anterior, el ESP32 ha podido solventar las limitaciones físicas de su antecesor, con distintas características que la hacen ideal para el desarrollo de una gran variedad de aplicaciones, en este apartado solo se hará referencia a los aplicados en el presente proyecto.

Sensores capacitivos

A diferencia de los botones tradicionales que requieren una conexión eléctrica directa y una presión física para cerrar un circuito, este tipo de sensores responden a la alteración del campo eléctrico cercano al sensor. Gracias a que cuenta con los principios de la capacitancia, la que se fundamenta en la capacidad de un objeto para almacenar carga eléctrica.
En estado de "reposo" (sin tocar), el pin que cuenta con esta característica, tiene una cierta capacitancia base, principalmente debido a las trazas de la PCB, los componentes cercanos y la capacitancia parásita. Cuando se acerca un dedo (que es conductor, aunque no perfecto) al pin táctil, se introduce otro objeto conductor en el campo eléctrico. este provoca dos efectos principales:

- Aumenta el área efectiva de las "placas" del capacitor.
- Introduce un nuevo dieléctrico (parcialmente).

Esto aumenta la capacitancia total del sistema, aunque el ESP32 no mida directamente esta capacitancia en faradios. Sino utiliza un método que se basa en el tiempo que tarda en cargar o descargar el capacitor formado por el pin táctil y el entorno, realizando estas mediciones de forma continua y compara el valor actual con un valor de referencia (la capacitancia base). Un cambio significativo (un aumento) indica que se ha detectado un "toque".

Modos Wifi

El ESP32 cuenta con distintos modos de conexión los cuales son:

- *Station (STA):* se conecta a una red Wi-Fi existente (como un teléfono actuando como hotspot o un router). Es el modo "cliente" tradicional.
- *Access Point (AP)*: crea su propia red Wi-Fi a la que otros dispositivos pueden conectarse directamente. Actúa como un pequeño router.

- *Station + Access Point (AP-STA):* puede funcionar simultáneamente como cliente (conectado a una red Wi-Fi) y como punto de acceso (permitiendo que otros dispositivos se conecten a él).

Como se verá más adelante el ESP32 se dispuso para que funcionara en modo servidor, donde su rol principal es escuchar las peticiones de otros dispositivos (clientes) y responder a ellas.
Por lo cual se configuró en Modo Punto de Acceso (Access Point - AP), permitiendo configurarse para crear su propia red Wi-Fi, permitiendo que otros dispositivos (como el teléfono o PC) se conectan directamente al ESP32 como si fuera un router, asignando una direcciones IP a los clientes que se conectan. También se pone en Modo Estación (Station - STA), con el fin de conectarse a una red Wifi existente, permitiendo que otros dispositivos de la misma red puedan comunicarse con este módulo a través de la misma IP.
Para el inicio del servidor, hará uso de la biblioteca WiFiServer.h (Usada en Arduino) para crear un servidor que escuche el el puerto 80 para HTTP (Un puerto es como una "puerta" lógica a través de la cual se envían y reciben datos en una conexión de red). Una vez realizada estas configuraciones para ponerlo en funcionamiento, el ESP32 se mantendrá en espera de que los clientes intenten conectarse a su dirección IP y al puerto específico, para establecer una conexión.
Una vez establecida la conexión entre ambos, el ESP32 espera recibir la petición del cliente. Esta petición sigue el protocolo HTTP donde puede ser de tipos, como GET que solicita información al servidor y POST que envía datos al servidor.
Cuando se analiza la petición recibida para determinar qué está solicitando el cliente, se ejecutará según lo establecido en el código o para enviar respuestas, la prepara para enviarla de vuelta a través de la conexión establecida siguiendo el protocolo HTTP, indicando con un 200 OK si todo salió bien o 404 Not Found si no se encontró el recurso solicitado.

Después de haber corroborado en distintas pruebas la transmisión de datos por medio de los módulos nRF24L01 ubicados en las dos placas (Arduino UNO y ESP32), se encontró que dejaba de funcionar en momentos determinados, después de hacer una investigación, se encontró que este problema podría ser solucionado añadiendo un capacitor de aluminio de 10uF

Diseño Final

Una vez cubiertos los detalles funcionales del proyecto, cumpliendo con los requisitos mencionados anteriormente, el diseño del prototipo quedó de la siguiente manera:

Protoboard Arduino Uno

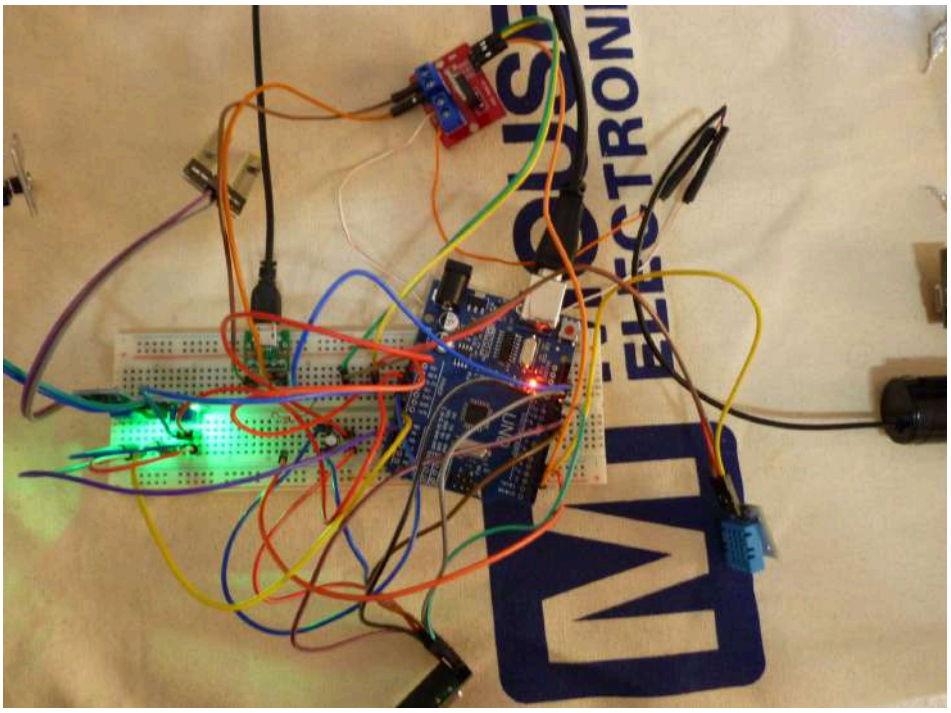


Figura 39: Conexiones Placa Arduino.
Fuente: Elaboración propia.

Como se muestra en la figura X, se ha puesto un módulo usb para suministrar 5V al MOSFET que se encargará de activar la bomba de agua, también un condensador de desacoplo de 10uF para que el módulo nRF24L01 no tenga inestabilidad a la hora de ejecutar sus tareas. Todas las conexiones fueron hechas usando cables dupont y diseñadas como se muestra a continuación:

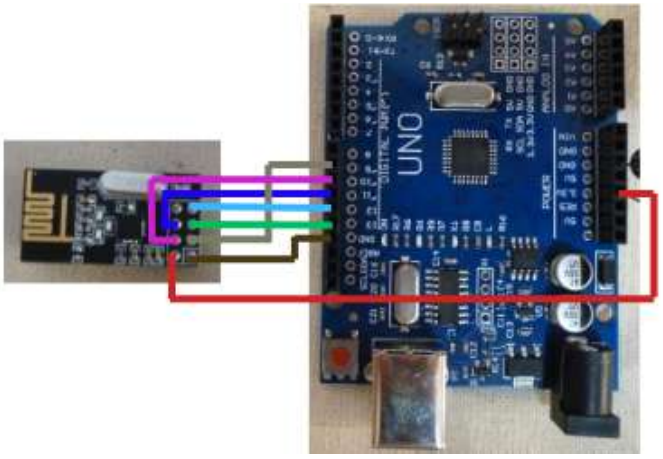


Figura 40: Conexión Arduino y módulo de comunicación.
Fuente: Elaboración propia.

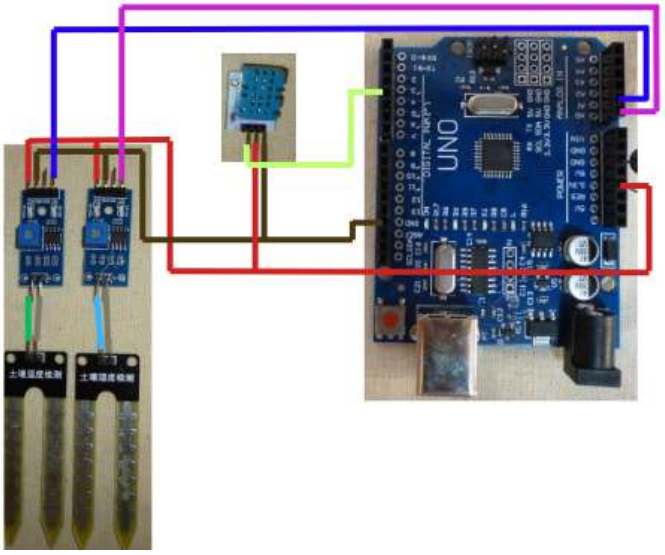


Figura 41: Conexión Arduino y Sensores.
Fuente: Elaboración propia.

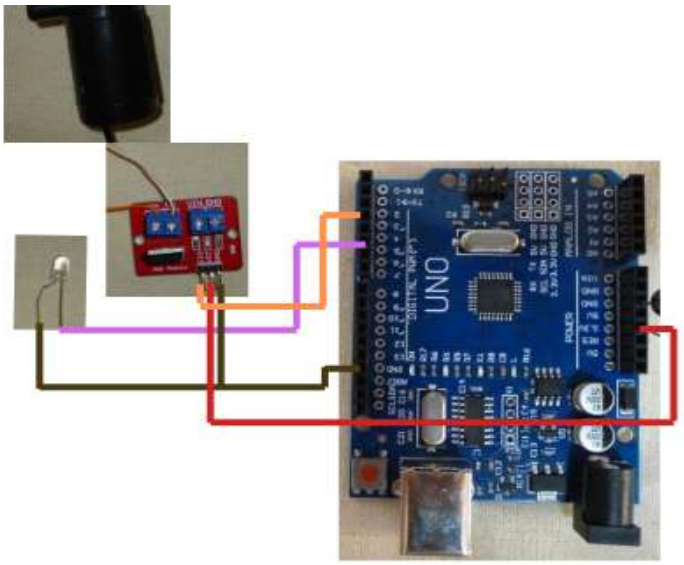


Figura 42: Conexión Arduino y Actuadores.
Fuente: Elaboración propia.

Protoboard ESP32

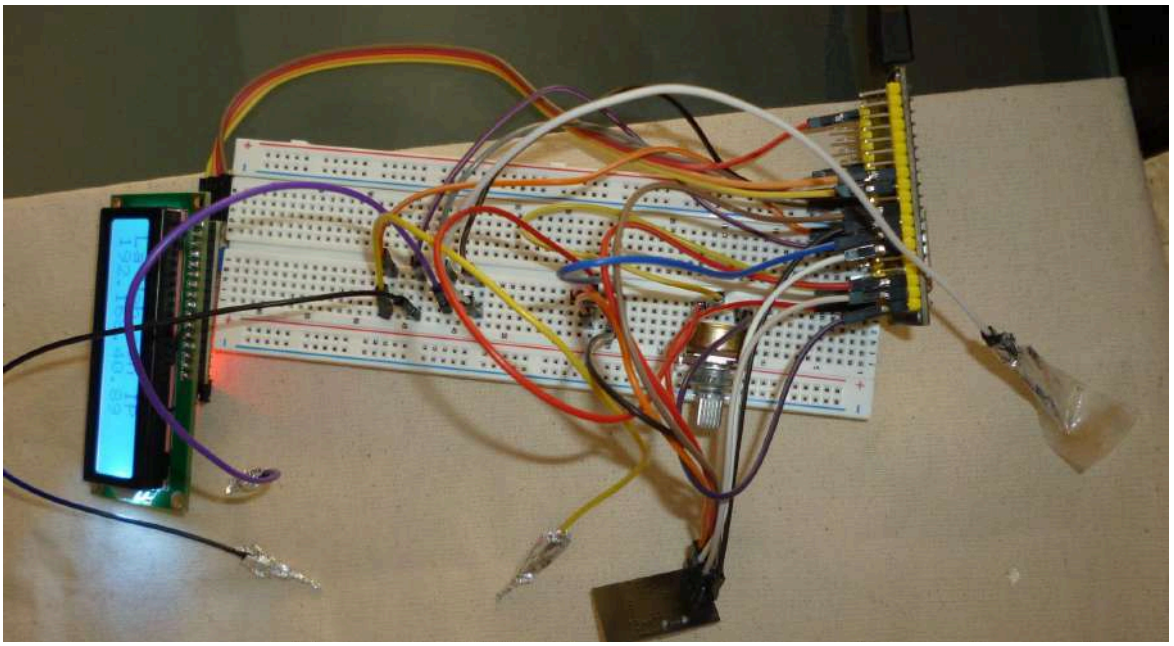


Figura 43: Conexiones Placa ESP32.
Fuente: Elaboración propia.

De igual manera que la configuración anterior, contará con un condensador de desacoplo de 10uF para que el módulo nRF24L01, también se realizó con papel de aluminio de un diámetro no determinado, para el funcionamiento de los botones de detección capacitiva (TouchPin). Todas las conexiones también fueron realizadas con cables dupont con la siguiente configuración:

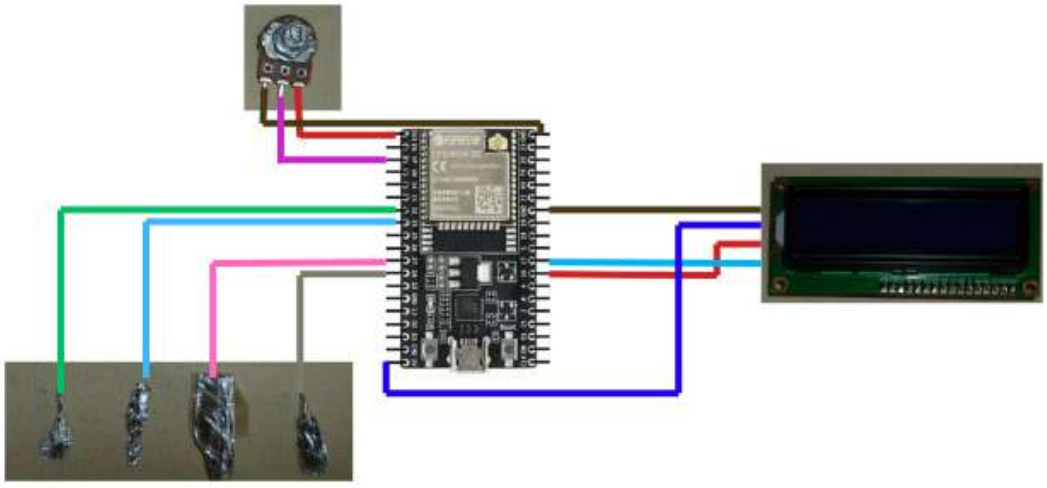


Figura 44: Conexiones Placa ESP32 e interfaz.
Fuente: Elaboración propia.

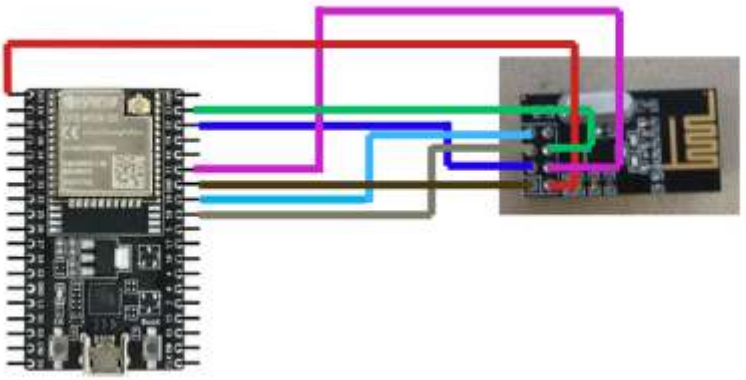


Figura 45: Conexiones Placa ESP32 y módulo de comunicación.
Fuente: Elaboración propia.

Pruebas

Con el desarrollo casi concluido del código utilizado en las dos placas (que será mostrado más adelante), se realizó distintas pruebas, primero con un potenciómetro que actuaría como las Sondas HL-69 en la placa donde se ubica el Arduino UNO para verificar si los datos transmitidos y su actuación era correcta.
Después también fueron puestas las dos sondas *HL-69* en tierra húmeda y seca para verificar su funcionamiento y la activación del MOSFET al no cumplir con el umbral recibido por el ESP32.

Para la verificación de que todo fuera correctamente según lo planteado, se dispuso principalmente del Monitor Serial del Arduino IDE con el fin de ver todos los datos al instante y de la pantalla LCD, que se presenta a continuación.

```
Lecturas - S1:2%, S2:3%, Aire:43.80%, Temp:22.30°C
Enviando - S1:2%, S2:3%, Aire:43.80%, Temp:22.30°C -> OK
Umbral recibido: 20%
Umbral recibido: 21%
Umbral recibido: 20%
Umbral recibido: 20%
Umbral recibido: 20%
Umbral recibido: 20%
Lecturas - S1:3%, S2:3%, Aire:44.00%, Temp:22.30°C
Enviando - S1:3%, S2:3%, Aire:44.00%, Temp:22.30°C -> OK
Umbral recibido: 20%
Umbral recibido: 20%
Umbral recibido: 20%
Umbral recibido: 21%
Umbral recibido: 20%
Lecturas - S1:2%, S2:3%, Aire:44.00%, Temp:22.30°C
Enviando - S1:2%, S2:3%, Aire:44.00%, Temp:22.30°C -> OK
Umbral recibido: 20%
Umbral recibido: 20%
Bomba DESACTIVADA (tiempo completado)
Umbral recibido: 20%
Bomba ACTIVADA por 10 segundos
Umbral recibido: 20%
Lecturas - S1:2%, S2:3%, Aire:44.20%, Temp:22.30°C
Enviando - S1:2%, S2:3%, Aire:44.20%, Temp:22.30°C -> OK
Umbral recibido: 20%
```

Figura 46: Datos enviados y recibidos de Arduino.
Fuente: Elaboración propia.

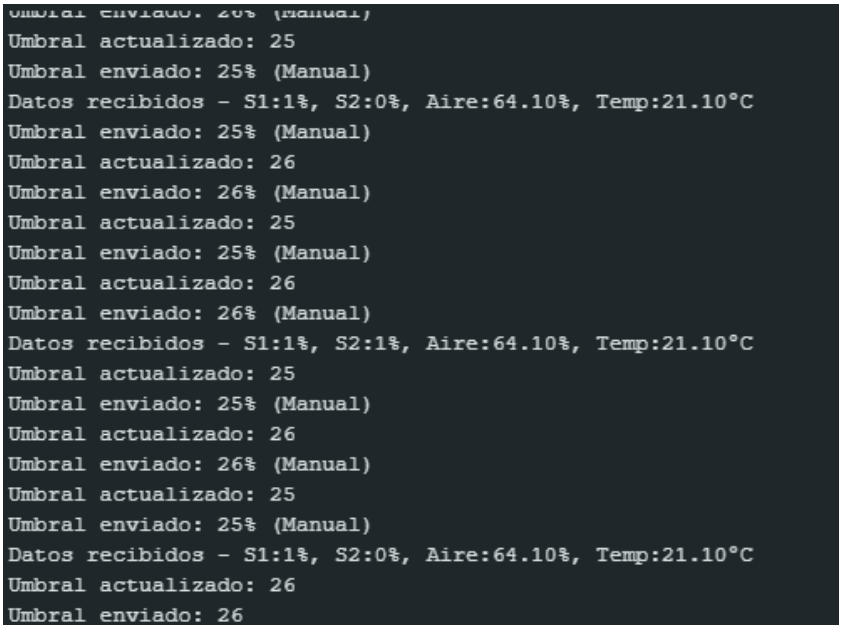


Figura 47: Datos enviados y recibidos de ESP32.
Fuente: Elaboración propia.

Figura 48: Datos de sensores en LCD del ESP32.
Fuente: Elaboración propia.

Figura 49: Datos de programa en LCD del ESP32.
Fuente: Elaboración propia.

También se realizó una prueba de distancia, con el fin de corroborar que la comunicación entre los dispositivos fuera la adecuada. Por lo cual, se estaciono la placa con el arduino en distintos puntos, con objetos de por medio.
Dando como resultado que su funcionamiento sería el óptimo, cuando se posicione hasta a *30 metros de distancia* entre las dos placas.

Diseño del Esquema Eléctrico y PCB

Tras haber elaborado el prototipo funcional y demostrarse cumplir con diversos requisitos mínimos, se procedió a la elaboración del diseño esquemático y enrutado de las pistas presentes en las placas que fueron realizados en la plataforma *EasyEDA*.

Placa Arduino UNO

El diseño elaborado para la Placa con el Arduino UNO, siguiendo el diseño elaborado en la fase de prototipos, dio como resultado el siguiente diagrama, donde fue dividido por bloques, según las funciones principales que tendrán.

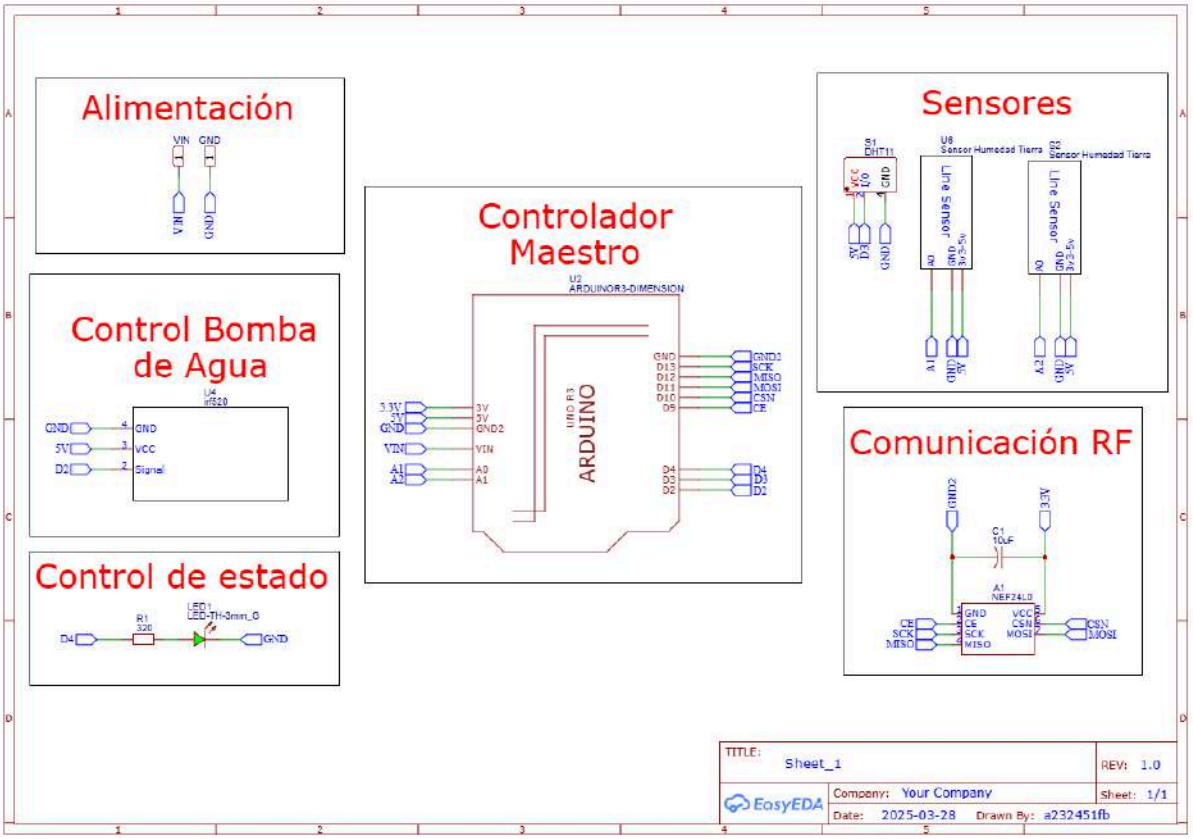


Figura 50: Esquema eléctrico de la placa Arduino.
Fuente: Elaboración propia.

El tamaño de la placa es de 100 milímetros de ancho y 66 milímetros de largo, donde la cara Bottom (inferior) se encontrará la mayoría de las pistas de ruteo con un diámetro de 1 milímetro y de 2 milímetros para la alimentación del exterior, contará también con un plano de masas que lo cubrirá en mayor parte, en cambio en la cara TOP (Superior) se encontrarán todos los componentes que serán ensamblados, como también ciertas conexiones que no fueron posibles realizarlas en la cara Bottom. El diseño fue el siguiente:

Cara TOP

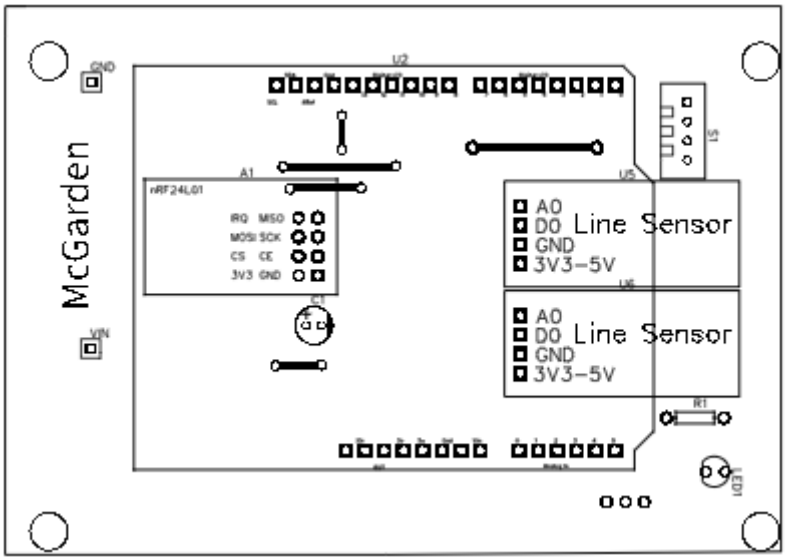


Figura 51: Cara Top Placa Arduino
Fuente: Elaboración propia.

Cara Bottom

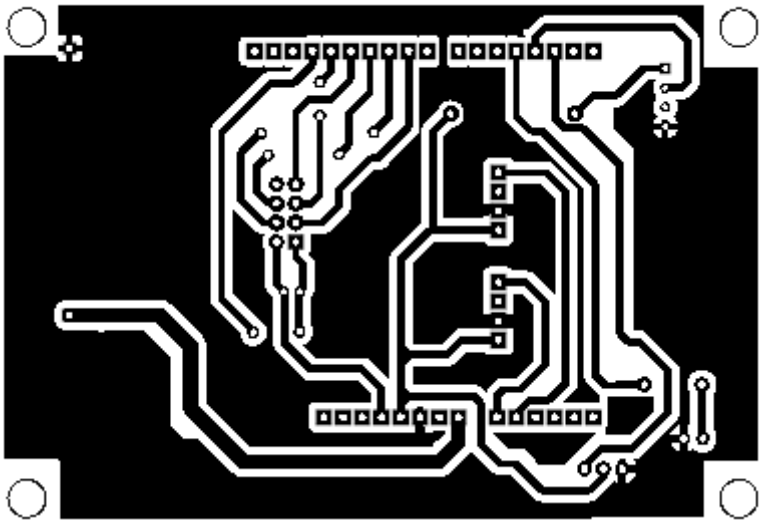


Figura 52: Cara Bottom Placa Arduino.
Fuente: Elaboración propia.

Placa ESP32

Como se mencionó en el diseño anterior, para la placa que contará con el ESP32, fue elaborado de la siguiente manera:

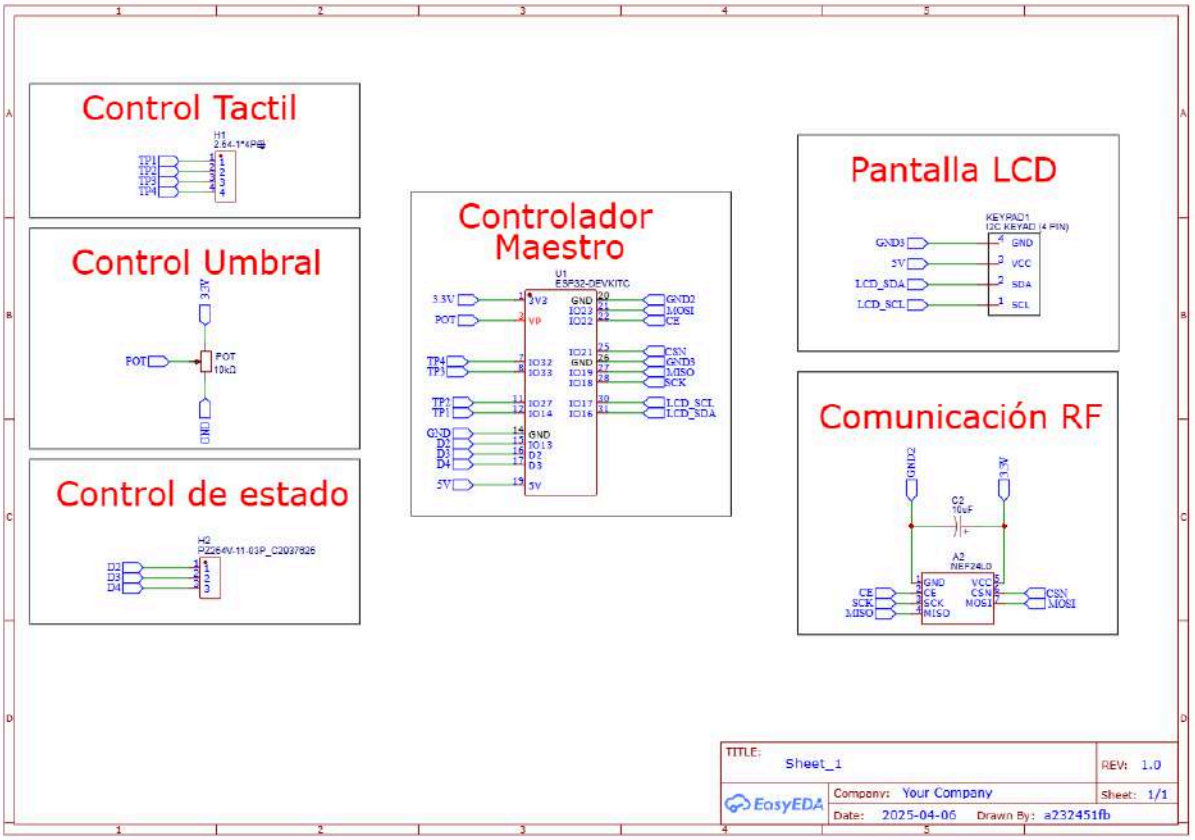


Figura 53: Esquema eléctrico de la placa ESP32.
Fuente: Elaboración propia.

El tamaño de la placa es de 72 milímetros de ancho y 70 milímetros de largo, siguiendo la misma configuración de la anterior placa realizada, aunque no contará con una ruta de alimentación externa, esta será suministrada directamente al ESP32. El diseño fue el siguiente:

Cara Top

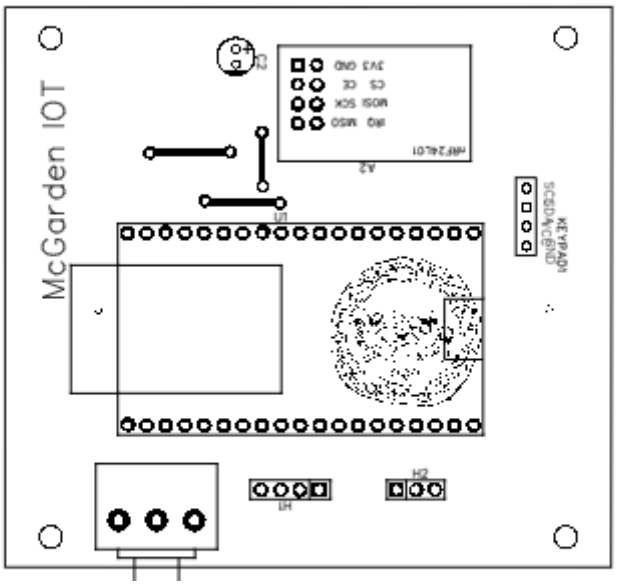


Figura 54: Cara Top Placa ESP32.
Fuente: Elaboración propia.

Cara Bottom

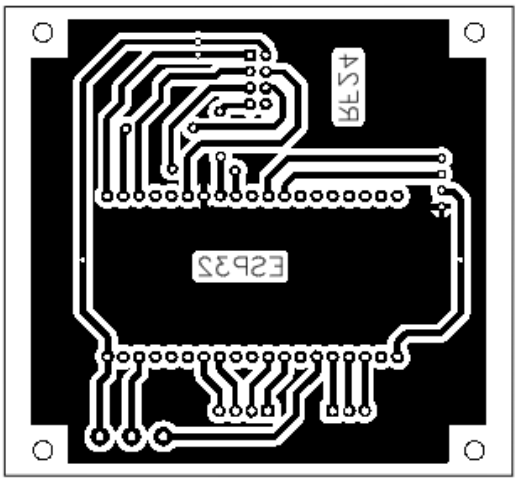


Figura 55: Cara Bottom Placa Arduino.
Fuente: Elaboración propia.

4.4. Construcción

Placas

Placa Arduino
Cumpliendo los requisitos mencionados anteriormente, se ha elaborado la soldadura y ensamble de los siguientes componentes:

Componente	**Cantidad**
Conectores de Pin Hembra	8
Conectores de Pin Macho	49
Condensador electrolítico de 10uF	1
Cableado electrico	1 metro
Modulo nRF24L01+	1
Modulo IRF520	1
Modulo LM393	2

Tabla 7: Componentes usados en la placa Arduino uno.
Fuente: Elaboración propia.

Después de haber realizado el proceso químico necesario para la impresión de las placas, ha dado el siguiente resultado, listo para empezar la soldadura de los componentes mencionados.

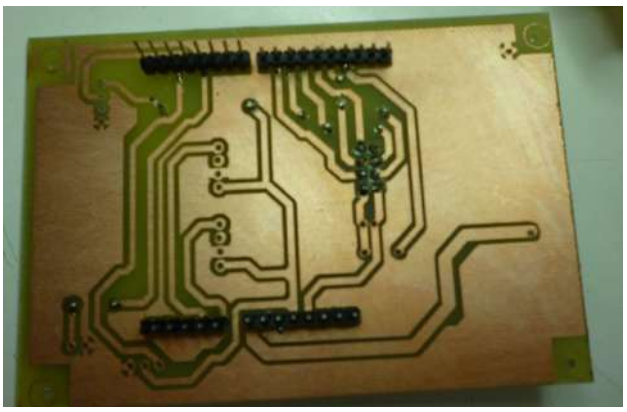


Figura 56: Impresión Placa Arduino con componentes.
Fuente: Elaboración propia.

Puesto todos los materiales y añadiendo los módulos que se hará cargo de controlar, este fue el resultado:

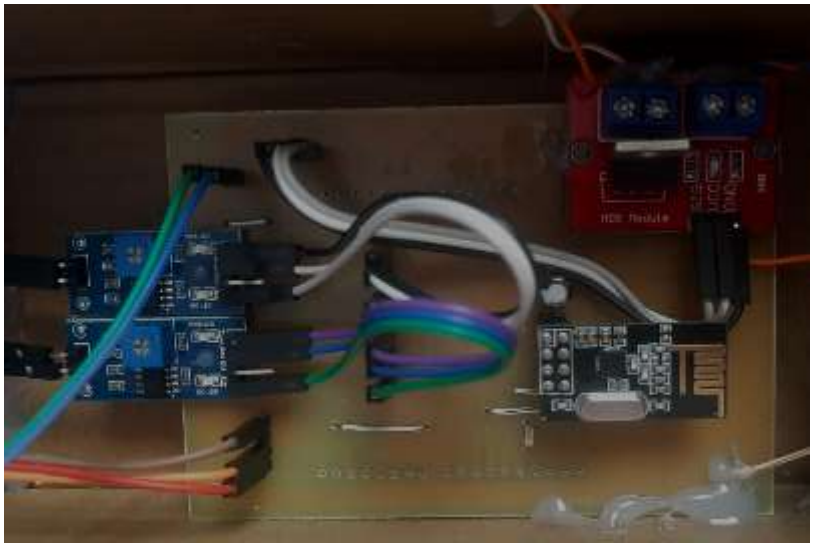


Figura 57: Diseño final Placa Arduino.
Fuente: Elaboración propia.

Como se aprecia en la figura 57, la conexión de todos los módulos serán hechas por medio de cables dupont a los pines machos puestos estratégicamente en diseño de la placa.

Placa ESP32

Como se ha mencionado anteriormente, el diseño de esta placa fue elaborado con los siguientes componentes:

Componente	**Cantidad**
Conectores de Pin Hembra	54
Conectores de Pin Macho en L	4

Condensador electrolítico de 10uF	1
Potenciometro	1
Cableado electrico	20 cm

Tabla 8: Componentes usados en la placa ESP32.
Fuente: Elaboración propia.

Ya que el diseño de la placa debía cumplir con el requisito de poder ser impreso en una placa de 100 mm x 160 mm, junto a la anterior realizada. fue elaborado en un solo proceso, dando como resultado:

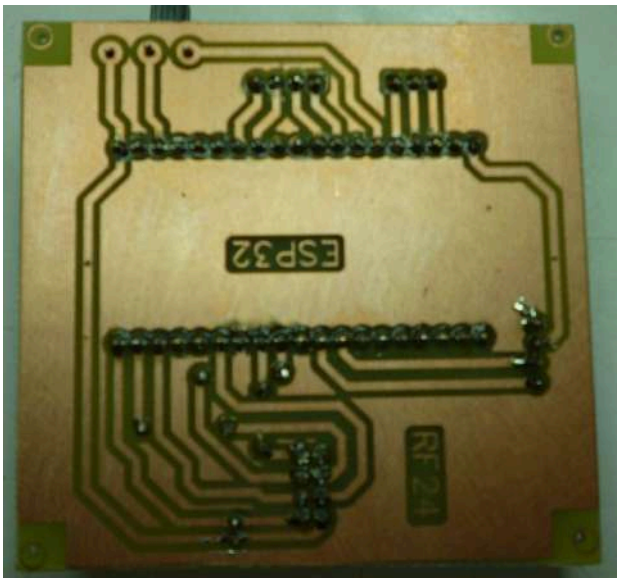


Figura 58: Impresión Placa ESP32.
Fuente: Elaboración propia.

Permitiendo integrar los componentes antes descritos de la siguiente manera:

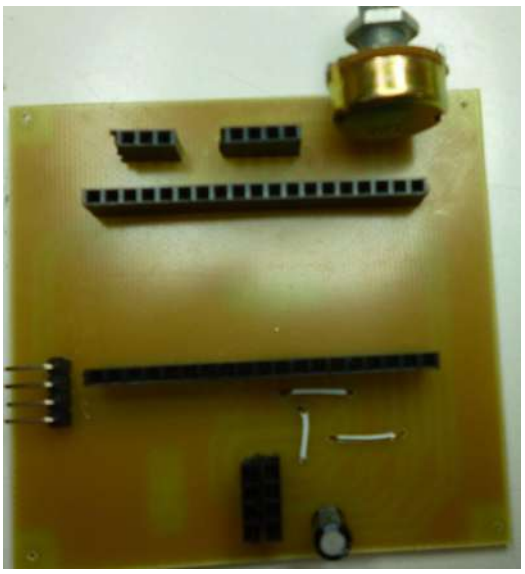


Figura 59: Diseño superior placa ESP32.
Fuente: Elaboración propia.

Como se observa en la figura 59, a diferencia de la placa anterior, esta fue diseñada con una arquitectura modular. Esto se debe a que una parte significativa de los componentes relacionados con la interfaz de usuario (Pantalla LCD y TouchPin) se integrará en el envoltorio del dispositivo, mientras que los componentes funcionales principales se han dispuesto para facilitar su reemplazo o modificación.

Envoltorio

De acuerdo con los lineamientos definidos en el apartado de requisitos, durante el desarrollo del proyecto se propuso la fabricación de los envoltorios para albergar las placas y demás componentes funcionales utilizando tecnología de impresión 3D. Sin embargo, ante la restricción de tiempo y la exigencia de un dominio profesional de las herramientas de diseño, se ha tomado la decisión de diseñar y construir estos envoltorios con cartón reutilizable, el cual incorpora una capa plástica para mitigar el riesgo de daños por salpicaduras de agua.
La elaboración final partió del diseño de bocetos claves, donde se plasmó lo que deseaba ser realizado.
En este punto, los envoltorios de la placa del Arduino Uno pasará a llamarse estación de activación y la del ESP32 se llamará Estación de control

<u>Bocetos</u>
Estacion de activación:

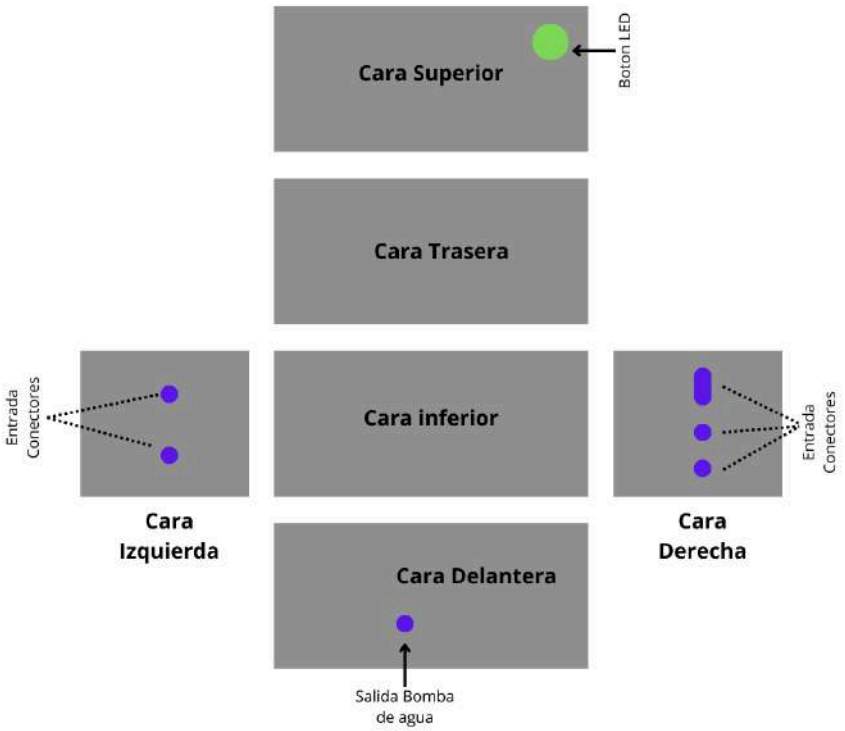


Figura 60: Boceto Estación de activación.
Fuente: Elaboración propia.

Estacion de Control:

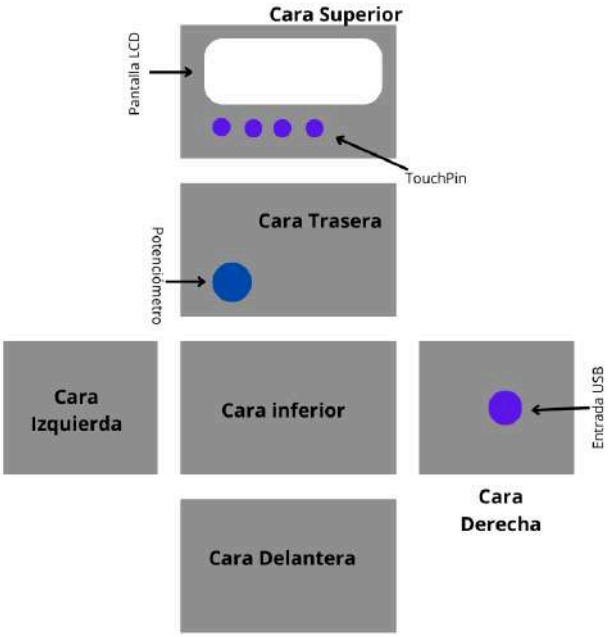
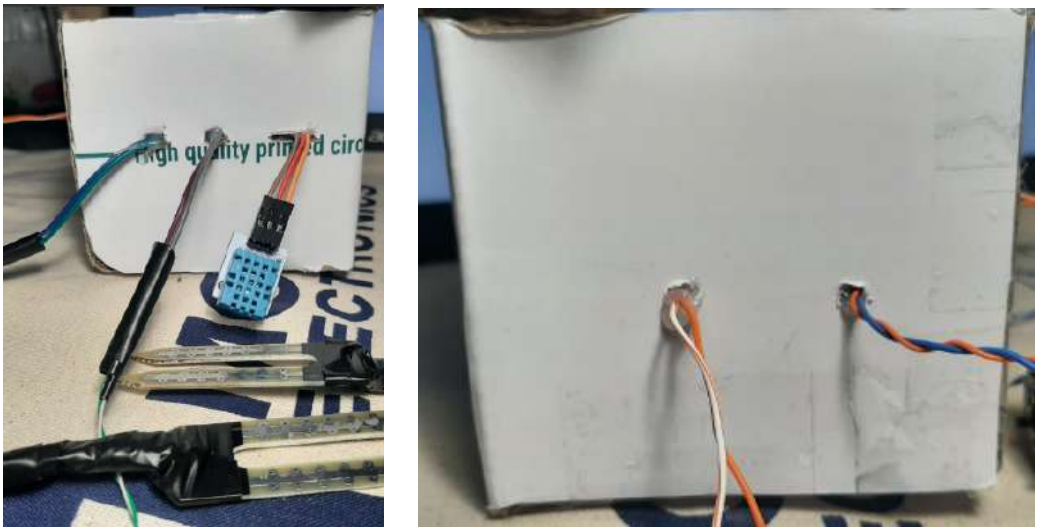


Figura 61: Boceto Estación de control.
Fuente: Elaboración propia.

Diseño Final

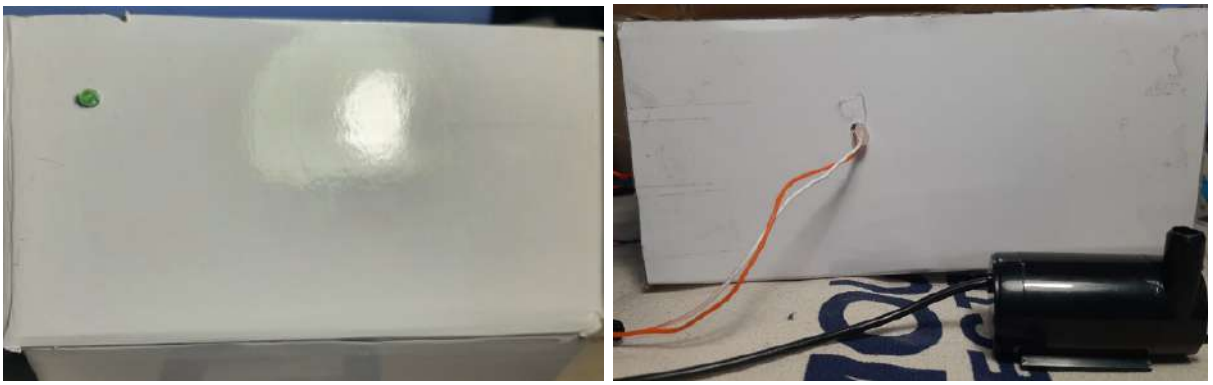
Siguiendo los bocetos elaborados, se ha realizado de tal forma sus envoltorios, junto con los conectores y demás elementos que lo componen.

Estación de activación



(a) Cara derecha *(b) Cara delantera*

Figura 62: Cara derecha e Izquierda, Estación de activación.
Fuente: Elaboración propia.



(a) Cara superior *(b) Cara delantera*

Figura 63: Cara superior y delantera, Estación de activación.
Fuente: Elaboración propia.

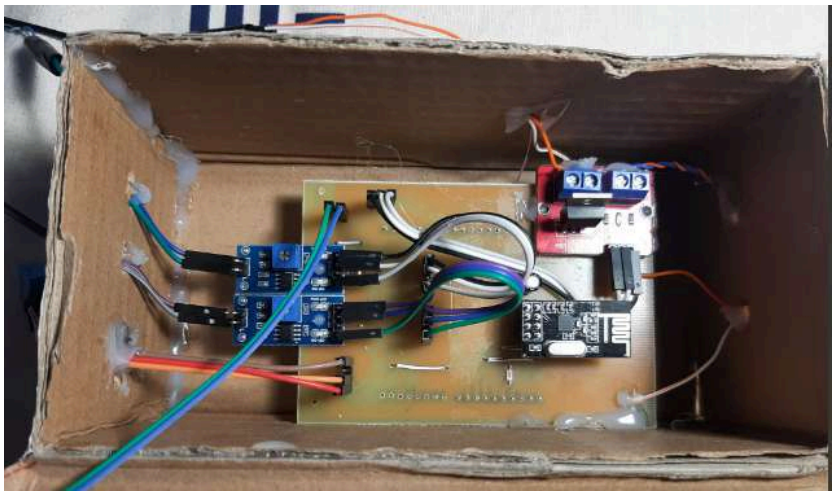


Figura 64: Interior inferior de la Estación de activación.
Fuente: Elaboración propia.

Figura 65: Interior superior de la Estación de activación.
Fuente: Elaboración propia.

Como se ha observado en las anteriores figuras de este apartado, el diseño ha sido de acorde con los bocetos antes mencionados. Todos los elementos que comunican con el exterior e interior de las estaciones, fueron puestos con silicona, además de crear con una placa de topos, el circuito para el correcto funcionamiento del LED que notificara la activación de la bomba de riego.

Estacion de Control

(a) Cara trasera *(b) Cara derecha*

Figura 66: Cara trasera y derecha, Estación de control.
Fuente: Elaboración propia.

Figura 67: Cara superior, Estación de control.
Fuente: Elaboración propia.

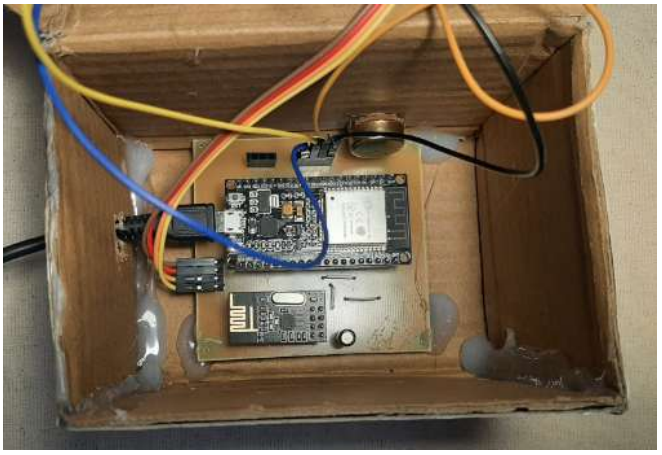


Figura 68: Interior inferior de la Estación de control.
Fuente: Elaboración propia.

Figura 69: Interior superior de la Estación de control.
Fuente: Elaboración propia.

Al igual que se mencionó anteriormente, los componentes del sistema de control se han fijado con silicona para facilitar su remoción en futuras actualizaciones. Como se observa en la figura 65, algunos conectores no se están utilizando debido a la inestabilidad que causaron durante pruebas de funcionamiento prolongadas. Además, el diseño de la entrada del conector USB en la cara derecha se realizó para asegurar la compatibilidad con una variedad de tamaños de conector.

Diseño Web

El diseño de la interfaz web se centró en la creación de una experiencia de usuario intuitiva y accesible, tanto en ordenadores de escritorio como en dispositivos móviles. Para lograr esto, se emplearon principios de diseño UX, como la organización clara de la información en bloques lógicos, el uso de íconos reconocibles para representar las diferentes funciones y un esquema de colores de alto contraste para mejorar la legibilidad.

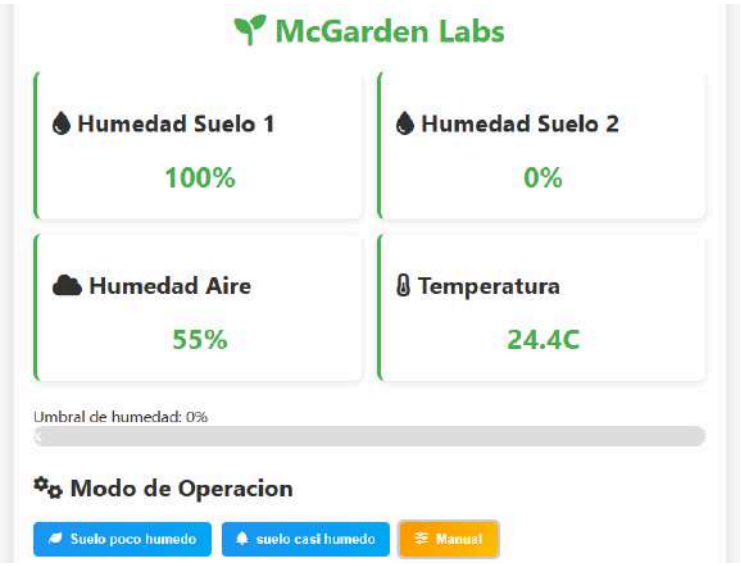


Figura 70: Interfaz Web de escritorio.
Fuente: Elaboración propia.

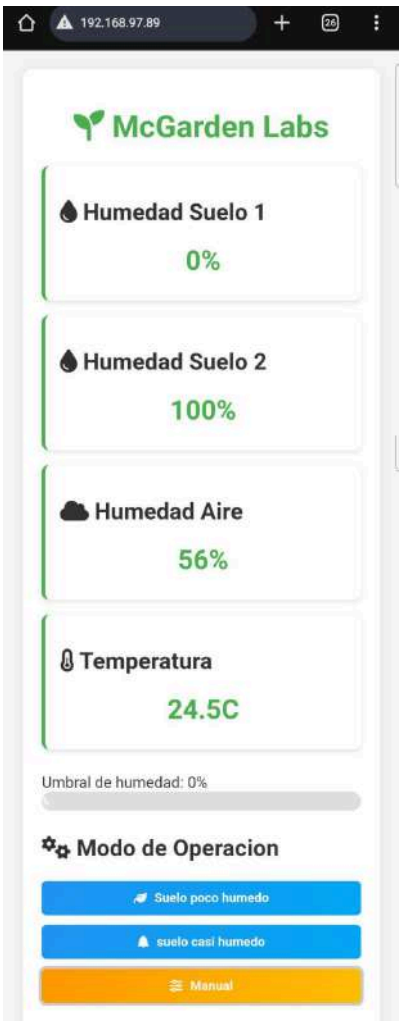


Figura 71: Interfaz Web de dispositivos móviles.
Fuente: Elaboración propia.

La interfaz se divide en dos bloques principales. El primero, el bloque de visualización de datos, muestra en tiempo real las lecturas de los sensores de la estación de activación, incluyendo la humedad del suelo (en %), la temperatura del aire (en °C) y la humedad ambiental (en %). Estos datos se actualizan dinámicamente cada 2 segundos mediante JavaScript, proporcionando al usuario una visión constante del estado del entorno.

Figura 72: Bloque de datos en interfaz Web.
Fuente: Elaboración propia.

El segundo bloque, el bloque de control, permite al usuario establecer los umbrales de humedad deseados para el riego automático y activar el riego manual si es necesario. Los botones para estas acciones proporcionan una respuesta visual inmediata al ser presionados, y se implementaron mensajes de confirmación para guiar al usuario a través del proceso.

Figura 73: Bloque de datos en interfaz Web.
Fuente: Elaboración propia.

La decisión de alojar la interfaz web directamente en el ESP32 se tomó para simplificar la arquitectura del sistema y reducir la dependencia de servidores externos, lo que mejora la velocidad de respuesta y la fiabilidad. El desarrollo del codigo utilizado, esta explicado en el siguiente apartado.

4.5. Programación

Código Arduino UNO

Librerias

Fueron incluidas las siguientes librerías en el desarrollo del código:

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <DHT.h>
```

- *<SPI.h>*: Utilizada para la comunicación entre el módulo NRF24L01+ y el Arduino, mediante el protocolo SPI (Serial Peripheral Interface) manejando la comunicación de bajo nivel.
- *<nRF24L01.h>*: Usada debido a que contiene las definiciones y funciones necesarias para controlar el chip de radio nRF24L01. permitiendo configurar el módulo como sea necesario.
- *<RF24.h>*: Basada en la librería anterior, proporciona una interfaz de programación de alto nivel, facilitando la configuración y el manejo de las comunicaciones del módulo, abstrayendo algunas de las complejidades de la comunicación SPI y los registros del chip.
- *<DHT.h>:* Es necesaria para interactuar con los sensores del módulo DHT11 y poder leer los datos de este sensor conectado al pin digital de nuestro arduino.

Pines

Las conexiones realizar con el arduino están puestas de tal forma:

```
#define CE_PIN   9
#define CSN_PIN 10
#define sensor1 A0
#define sensor2 A1
#define Mosfet_PIN 2
#define Led_Pin 4
#define DHTPIN 3
#define DHTTYPE DHT11
```

- *CE_PIN 9 y CSN_PIN 10*: Pines utilizados para la configuración del nRF24L01, donde CE (Chip Enable) es para activar o desactivar la comunicación del módulo y CSN (Chip Select Not) encargado de seleccionar este chip, cuando hay distintos dispositivos SPI en el mismo bus.
- *sensor1 A0 y sensor2 A1*: Puestos para recibir los valores analógicos de las sondas de humedad HL-69.

- *Mosfet_PIN 2:* Trabaja como interruptor el pin para activar o desactivar el Gate(G) del modulo MOSFET.
- Led_Pin 4: Se encarga de enseñar cuando se ha activado el Mosfet.
- *DHTPIN 3*: Indica que pin recibe los datos del sensor DHT11.
- *DHTTYPE DHT11*: Le indica a la librería DHT.h que tipo específico de sensor DHT se esta usando.

Variables Globales

```
const byte slaveAddress[5] = {'R','x','A','A','A'};
const byte thisMasterAddress[5] = {'T','x','A','A','A'};
```

Constantes de 8 bits, encargadas de asignar mediante una array de 5 elementos, las direcciones asignadas para la comunicación del módulo nRF24L01+

```
RF24 radio(CE_PIN, CSN_PIN);
DHT dht(DHTPIN, DHTTYPE);
```

- *RF24 radio(...):* Indica a la librería RF24 que pines han sido puestos para el CE y el CSN, para la comunicación física.
- *DHT dht(...):* Similar a la anterior, indicando a la librería dht que pin y qué clase de sensor fue puesto.

```
struct SensorData {
  int humedadSuelo1;
  int humedadSuelo2;
  float temperatura;
  float humedadAire;
};

char dataReceived[32] = {0};
int umbralRecibido = 0;
bool nuevoUmbral = false;
```

- struct SensorData { ... }: define una nueva estructura de datos, para agrupar los distintos datos de los sensores antes de ser enviados a través de la radio. En lugar de enviar cada valor por separado. Facilitando el envío y la recepción de todos los datos.
- *char data…; int umbral…, y bool nuevo…:* estas tres líneas se encargan de almacenar los datos de control recibidos, iniciar el valor en 0 e indicar cuando se ha recibido un nuevo valor para que lo gestione, respectivamente.

```
unsigned long currentMillis;
unsigned long prevSensorRead = 0;
```

```
const unsigned long sensorInterval = 2000;
unsigned long prevTxMillis = 0;
const unsigned long txInterval = 2000;
unsigned long MosfetStartTime = 0;
const unsigned long MosfetDuracion = 10000;
bool MosfetActiva = false;
```

Estas variables se se utilizan para gestionar el tiempo dentro del bucle principal (loop()) del Arduino, permitiendo que ciertas acciones (como la lectura de sensores, la transmisión de datos y el control del MOSFET) se realicen a intervalos específicos y durante duraciones determinadas, sin usar la función delay(), que detendría la ejecución de todo el programa.

```
int lastHumedadSuelo1 = 0;
int lastHumedadSuelo2 = 0;
float lastTemperatura = 0;
float lastHumedadAire = 0;
```

Estas 4 variables actúan como un **almacén temporal** para los valores más recientes leídos de cada uno de los sensores.

setup

La configuración de inicialización será la siguiente:

```
void setup() {
    Serial.begin(115200);
    while (!Serial);
    Serial.println("Iniciando Maestro con DHT11");

    dht.begin();

    if (!radio.begin()) {
        Serial.println("Error: Módulo RF24 no detectado!");
        while (1);
    }
```

Este bloque se encarga de iniciar la comunicación serial (UART) entre el Arduino y la computadora, como también de indicar si ha iniciado con éxito el módulo nRF24L01 y el sensor DHT11. Al contar con while(1), crea un bucle infinito cuya condición es verdadera, permitiendo que continúe sin generar un atasco.

```
    radio.setChannel(108);
    radio.setDataRate(RF24_250KBPS);
    radio.setRetries(5, 15);
```

```
    radio.setPALevel(RF24_PA_LOW);
    radio.openWritingPipe(slaveAddress);
    radio.openReadingPipe(1, thisMasterAddress);
    radio.startListening();
```

En estas líneas se configuran los parámetros claves para el funcionamiento del nRF24L01 (deberá ser igual en el otro arduino), como lo son:

- *radio.setChannel(108):* establecido el canal de comunicación 108 (frecuencia central = 2400 + 108 MHz) para evitar interferencias con Wifi.
- *radio.setDataRate(RF24_250KBPS)*: Configura la velocidad de transmisión con mayor alcance.
- *radio.setRetries(5, 15):* se establece la configuración de reintentos automáticos si falla la comunicación, siendo de 15 intentos con un delay de 5 (en múltiplos de 250µs → 5×250µs = 1.25ms) entre intentos.
- *radio.setPALevel(RF24_PA_LOW)*: Ajusta la potencia de transmisión a baja (-18dBm), permitiendo un menor consumo energético.
- *radio.openWritingPipe(slaveAddress)* y *radio.openReadingPipe(1,thisMasterAddres):* según la variable designada, establece las direcciones de envío y recepción de datos.
- *radio.startListening():* pone el módulo en modo receptor al iniciar.

```
    pinMode(Mosfet_PIN, OUTPUT);
    digitalWrite(Mosfet_PIN, LOW);
    pinMode(Led_Pin, OUTPUT);
    digitalWrite(Led_Pin, LOW);
    leerSensores();
}
```

Complementando la configuración inicial de la placa arduino, estas 3 líneas preparan el funcionamiento del módulo MOSFET, encargado de activar la bomba de agua, al cual se le ha designado el Pin 2 y comenzará siempre inactivo para evitar activaciones falsas, como la asignación del Led_Pin para que actúe en conjunto.
También activa la función leerSensores con el fin de iniciar las variables dentro de estas, evitando enviar valores basuras en el primer ciclo del loop.

LOOP

```
void loop() {
    currentMillis = millis();

    if (currentMillis - prevSensorRead >= sensorInterval) {
        prevSensorRead = currentMillis;
        leerSensores();
    }
```

```
    if (currentMillis - prevTxMillis >= txInterval) {
        prevTxMillis = currentMillis;
        enviarDatosSensores();
    }
```

Inicia convirtiendo el valor de *millis()* en la variable *currentMillis,* utilizada para trabajar en los dos siguientes bloques:

- *if (currentMillis - prevSensorRead >= sensorInterval) { ... }:* Este bloque controla si ha pasado suficiente tiempo para leer los sensores nuevamente. Calculando la diferencia de tiempo entre el momento actual y la última vez que se leyeron los sensores y mediante >= *sensorInterval,* comparar esta diferencia de tiempo con el intervalo deseado entre lecturas de sensores.
  Si se ha cumplido esta condición, *prevSensorRead,* pasará a guardar el valor actual del *currentMillis* (Esto guarda el momento actual como la última vez que se leyeron los sensores, para la próxima vez que se realice la verificación del intervalo). También llamará a la función *leerSensores(),* para que se ejecute.
- *if (currentMillis - prevTxMillis >= txInterval) { ... }:* Trabaja igual que el anterior, con el tiempo designado en las variables globales, pero con la caracteristica de que sera el encargado de activar la función *enviarDatosSensores(),* que veremos mas adelante.

```
    recibirDatosControl();
    controlBomba();
```

Para cerrar nuestro loop, también se llamará a ejecutar estas funciones, que serán descritas más adelante.

Función leerSensores

```
void leerSensores() {
    // Humedad del suelo
    int sumaH1 = 0, sumaH2 = 0;
    const int muestras = 5;

    for(int i = 0; i < muestras; i++) {
        sumaH1 += analogRead(sensor1);
        sumaH2 += analogRead(sensor2);
        delay(50);
    }

    lastHumedadSuelo1 = map(sumaH1/muestras, 0, 1023, 100, 0);
```

```
    lastHumedadSuelo2 = map(sumaH2/muestras, 0, 1023, 100, 0);
```

Como se ha señalado anteriormente, se diseño la función *leerSensores,* para leer los sensores conectados al arduino. esta funciona de tal manera:

- *int sumaH1 = 0, sumaH2 = 0:* Son declaras para guardar las lecturas analógicas de los sensores sensores del suelo.
- *const int muestras = 5:* Se declara un número de lecturas que tomará cada sensor para luego calcular un promedio. Al tomar múltiples lecturas y promediarlas ayudará a reducir el ruido y obtener una lectura más estable.
- *for(int i = 0; i < muestras; i++) { ... }:* Este bucle se ejecutará mientras "i" sea menor que el número de muestras(5), mientras realiza las lecturas de los sensores y asignándoles las variables presentes dichas, contando con un delay de 50 ms para estabilizarse y evita la lectura de valores demasiado rápidos que podrían ser inconsistentes.
- *lastHumedadSuelo1 = map… y lastHumedadSuelo2 = map…:* Calculan el promedio de las 5 lecturas tomadas por los sensores y mediante la función *map* Mapea el valor promedio del rango analógico (0-1023) a un rango de porcentaje(0-100), asumiendo una relación inversa entre el valor analógico y la humedad.para que sean utilizados como ya hemos visto.

```
    float h = dht.readHumidity();
    float t = dht.readTemperature();

    if (!isnan(h)) { lastHumedadAire = h; }
    if (!isnan(t)) { lastTemperatura = t; }
```

- *float h = dht.readHumidity() y float t = dht.readTemperature()*: Se encargan de almacenar mediante variables, los datos de humedad y temperatura suministrados por el sensor DHT usando las funciones *dht.read* que esta en su libreria.
- *if (!isnan(h)) {...}:* (Is Not a Number) es una función diseñada para verificar siempre que los datos suministrados por el sensor sean los esperados.

```
    Serial.print("Lecturas - S1:");
    Serial.print(lastHumedadSuelo1);
    Serial.print("%, S2:");
    Serial.print(lastHumedadSuelo2);
    Serial.print("%, Aire:");
    Serial.print(lastHumedadAire);
    Serial.print("%, Temp:");
    Serial.print(lastTemperatura);
    Serial.println("°C");
```

Este bloque es el encargado de mostrar los valores tomados por los sensores. Se dispone para verificar si coincide con los datos enviados al otro módulo RF24.

Función enviarDatosSensores

```
void enviarDatosSensores() {
    radio.stopListening();
    delay(10);
```

Como se ha mostrado anteriormente, esta función será llamada después de la recepción de datos, por lo cual, inicia desactivando la lectura con una pausa de 10 ms para proceder con el envío de datos.

```
    SensorData datos;
    datos.humedadSuelo1 = lastHumedadSuelo1;
    datos.humedadSuelo2 = lastHumedadSuelo2;
    datos.temperatura = lastTemperatura;
    datos.humedadAire = lastHumedadAire;
```

Se declara una variable llamada datos de tipo SensorData (estructura creada en las variables globales) para asignar los valores de las últimas lecturas de los sensores, a los miembros de la estructura datos. Agrupando en una solo variable para ser enviada.

```
    bool resultado = radio.write(&datos, sizeof(datos));
```

Esta línea de código toma la estructura *datos* que contiene las lecturas de los sensores, obtiene su dirección de memoria y su tamaño, y luego utiliza la función *radio.write()* para intentar enviar estos datos de forma inalámbrica. El éxito o el fracaso de esta transmisión se guarda en la variable *resultado*.

```
    Serial.print("Enviando - S1:");
    Serial.print(datos.humedadSuelo1);
    Serial.print("%, S2:");
    Serial.print(datos.humedadSuelo2);
    Serial.print("%, Aire:");
    Serial.print(datos.humedadAire);
    Serial.print("%, Temp:");
    Serial.print(datos.temperatura);
    Serial.println(resultado ? "°C -> OK" : "°C -> Fallo");
       radio.startListening();
}
```

En estas últimas líneas de la función se encargan de mostrar en el *Monitor Serial* si los datos han sido enviados con éxito y cuales son, como también indicará si ha fallado la transmisión.
La función cierra indicando que se prepare de nuevo a recibir datos desde el otro módulo nRF24L01 en el siguiente ciclo del *loop*.

Función recibirDatosControl

```
void recibirDatosControl() {
    if (radio.available()) {
        memset(dataReceived, 0, sizeof(dataReceived));
        radio.read(&dataReceived, sizeof(dataReceived))
```

Empieza con un bloque de control que verifica si el módulo de nRF24L01 ha recibido algún dato que aún no ha sido leído por el arduino, por medio del *radio.available()*, devuelve true si hay datos disponibles en el buffer de recepción del módulo de radio y false si no es el caso.
En el caso de ser *True,* se activa la función *memset,* que se encarga de llenar todo el array con valor 0 (borrando los datos anteriormente recibidos), preparando para recibir los nuevos valores, tambien se activara la función *radio.read(),* encargada de escribir losen la array *dataReceived.* Colocando *sizeof(dataReceived),* especificará el número máximo de bytes que tendrá que leer, determinado por la array *dataReceived.*

```
        if (strncmp(dataReceived, "u:", 2) == 0) {
            umbralRecibido = atoi(dataReceived + 2);
            nuevoUmbral = true;
```

En el bloque *if* anterior mencionado, tambien activara esta función en el caso de ser *true.*
Donde el bloque *if (strncmp(...),* verificará si los datos recibidos comienzan con la cadena de caracteres “u:” y comparando los primeros 2 caracteres de ambas cadenas, devolviendo 0 si ambos cadenas son iguales.
Al ser ejecutada, la línea *umbralRecibido = atoi(...)*, convierte la cadena de texto a partir del tercer carácter de *dataReceived* a un valor entero, que representa el nuevo umbral de humedad, que será usado para controlar la bomba de agua. También cuenta con la variable *nuevoUmbral,* la cual se usará para indicar por medio de un *true*, que ha sido recibido un nuevo dato que podrá ser tratado en la siguiente funcion (*controlBomba())* y actúe en consecuencia.

```
            Serial.print("Umbral recibido: ");
            Serial.print(umbralRecibido);
            Serial.println("%");
```

Este bloque se cierra visualizando en la pantalla los datos recibidos para conocer si coinciden con lo enviado. Cerrando también la función

Función controlBomba

```
void controlBomba() {
    if (nuevoUmbral) {
        nuevoUmbral = false;

        // Usar las variables renombradas CORRECTAMENTE
```

```
            bool debeActivar = (umbralRecibido > lastHumedadSuelo1) ||
(umbralRecibido > lastHumedadSuelo2);
```

Comienza con el bloque *if (nuevoUmbral) { ... },* verificando si ha sido recibido un nuevo Umbral de humedad por el estado de la variable *nuevoUmbral,* ejecutando inmediatamente la siguiente línea, si se ha recibido un nuevo valor, pasando luego a ejecutar *bool debeActivar…* declarando la variable *debeActivar*, que será verdadera si el umbral de humedad es mayor al último valor de humedad de suelo de por lo menos uno de los dos sensores.

```
        if (debeActivar && !MosfetActiva) {
            digitalWrite(Mosfet_PIN, HIGH);
            digitalWrite(Led_Pin, HIGH);
            MosfetStartTime = currentMillis;
            MosfetActiva = true;
            Serial.println("Bomba ACTIVADA por 10 segundos");
        }
    }
```

Después de haber realizado la verificación, este bloque if, será activado si la variable booleana *debeActivar* y *MosfetActiva* (invertida a true por medio del !) son true y procederá a encender el Pin que es el encargado de activar el MOSFET.
La línea *MosfetStartTime = currentMillis* se encargará de registrar el momento en que se activó el MOSFET y *MosfetActiva = true* de su estado actual.

```
         if (MosfetActiva && (currentMillis - MosfetStartTime >=
MosfetDuracion)) {
        digitalWrite(Mosfet_PIN, LOW);
        digitalWrite(Led_Pin, LOW);
        MosfetActiva = false;
        Serial.println("Bomba DESACTIVADA (tiempo completado)");
    }
```

Para cerrar la función, el bloque if, se encargará de desactivar el MOSFET cuando pase un tiempo determinado en su variable global; por lo cual, verificará si *MosfetActiva* es true y calculando la diferencia de tiempo entre el momento actual y el momento que se activó el MOSFET, los comparara con la duración que se desea que esté activa la bomba de agua (10 segundos). Si ha cumplido el tiempo activo, será apagado el Pin, como también puesto en false la variable *MosfetActiva* hasta su próximo ciclo hasta que sea requerida su activación.

El código completo se encontrará en el Anexo 12.
.

Código ESP32

Librerias

```
#include <SPI.h>
#include <RF24.h>
#include <nRF24L01.h>
#include <WiFi.h>
#include <WebServer.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
```

- *<SPI.h>, <RF24.h>, <nRF24L01.h>:* Utilizadas como ya se explicó anteriormente.
- *<WiFi.h>:* Utilizada para habilitar la conectividad WiFi en el ESP32. Permitiendo conectar el dispositivo a redes WiFi, obtener una dirección IP y establecer las bases para la comunicación a través de la red inalámbrica.
- *<WebServer.h>:* Permite que el ESP32 funcione como un servidor web, capaz de recibir peticiones HTTP de clientes (navegadores web) y enviar respuestas, como páginas HTML o datos. Facilita la creación de una interfaz web para interactuar con el dispositivo.
- *<Wire.h>:* Utilizada para la comunicación con dispositivos que utilizan el protocolo I2C (Inter-Integrated Circuit), como la pantalla LCD I2C. Proporciona las funciones necesarias para enviar y recibir datos a través de los pines SDA y SCL.
- *<LiquidCrystal_I2C.h>:* Librería específica para controlar pantallas LCD que utilizan una interfaz I2C. Simplifica la interacción con estas pantallas, permitiendo mostrar texto y otros elementos sin la necesidad de múltiples pines de conexión directa.

Pines

Todas las conexiones fueron puestas de tal forma:

```
#define CE_PIN 22
#define CSN_PIN 21
#define thresholdPin 36    // Potenciómetro
#define TOUCH_PIN_1 14     // Touch Pin para 30%
#define TOUCH_PIN_2 27    // Touch Pin para 70%
#define TOUCH_PIN_MAN 32   // Touch Pin para modo manual
#define touchNav_Pin 33   //Touch Pin Navegar por LCD
#define LCD_SDA 16
#define LCD_SCL 17
```

- *CE_PIN 9 y CSN_PIN 10*: Pines utilizados para la configuración del nRF24L01, donde CE (Chip Enable) es para activar o desactivar la

comunicación del módulo y CSN (Chip Select Not) encargado de seleccionar este chip, cuando hay distintos dispositivos SPI en el mismo bus.

- *thresholdPin:* Encargado de tomar los valores analógicos del potenciómetro.
- *TOUCH_PIN_1 14, TOUCH_PIN_2 27, TOUCH_PIN_MAN 32 y TouchNav_Pin 33:* Partiendo de las bases de los botones capacitivos que ofrece el ESP32, se realizó dichas conexiones para programar los umbrales y la navegación por la pantalla LCD.
- *LCD_SDA 16 y LCD_SCL 17:* Encargados de la comunicación con el módulo I2C para el funcionamiento de la pantalla LCD.

Configuración inicial

```
const char* ssid = "McMarciano";
const char* password = "McMa1503";
```

Se definen las credenciales de la red WiFi a la que el ESP32 intentará conectarse. La variable ssid almacena el nombre de la red ("McMarciano") y la variable password contiene la contraseña ("McMa1503").

```
const byte thisSlaveAddress[5] = {'R','x','A','A','A'};
const byte masterAddress[5] = {'T','x','A','A','A'};
```

Constantes de 8 bits, encargadas de asignar mediante una array de 5 elementos, las direcciones asignadas para la comunicación entre los módulos nRF24L01+, estas son iguales que las del código anterior.

```
LiquidCrystal_I2C lcd(0x27,16,2);
```

Crea una instancia del objeto lcd de la librería LiquidCrystal_I2C. especificando la dirección I2C, el número de columnas y de número de filas. Esto permite controlar la pantalla LCD conectada mediante el protocolo I2C.

```
RF24 radio(CE_PIN, CSN_PIN);
```

Indica a la librería RF24 que pines han sido puestos para el CE y el CSN, para la comunicación física.

```
WebServer server(80);
```

Crea una instancia del objeto server de la librería WebServer. El argumento 80 especifica el puerto HTTP estándar en el que el servidor web escuchará las peticiones entrantes de los clientes.

Variables Globales

```
int potValue = 0;
int umbral = 0;
int touchThreshold = 35;
bool manualMode = true;
char dataToSend[32] = "ACK-Eslavo";
unsigned long lastTouchTime = 0;
```

```
const long touchDebounce = 500;
int lcdScreen = 0;
bool buttonReleased = true;
```

- *int potValue…:* Declarada para almacenar las lecturas analógicas del *thresholdPin*.
- *int umbral…:* Declarada para almacenar el valor del umbral de humedad.
- *int touchThreshold…:* Declarada en 35 para ajustar el nivel de sensibilidad para la detección de los pines táctiles.
- *bool manualMode…:* Variable encargada de controlar el modo de operación del sistema.
- *char dataToSend[32] …:* Declara un array de caracteres con una capacidad de 32 elementos. Esta variable se utilizará como un buffer para almacenar los datos que se enviarán a través de la comunicación por radio RF24.
- *unsigned long lastTouchTime… y const long touchDebounce…:* Puestas para controlar el tiempo de rebote que sera usado para los TouchPin.
- *int lcdScreen… y bool buttonReleased…:* Encargadas de controlar y gestionar la visualización de la pantalla LCD.

```
struct SensorData
  {
    int16_t humedadSuelo1;
    int16_t humedadSuelo2;
    float temperatura;
    float humedadAire;
  };
SensorData newData;
```

Al igual que fue descrita en el código anterior, esta estructura se encarga de gestionar los datos que han sido recibidos por el módulo de radio. Se ha declarado las variables a *int16* debido a que el ESP32 trabaja los datos suministrados a este rango.

```
// Funciones
void handleRoot();
void handleSetMode();
void handleGetData();
void enviarUmbral();
void getData();
void botonNavegador();
void initWiFi();
void checkTouchPins();
void ActualizarUmbral();
```

```
void ActualizacionLCD();
String getWebPage();
```

Se ha declarado las funciones que serán usadas dentro de todo el código, asegurando la disponibilidad de la función y evitando errores de compilación.

setup

La configuración de inicialización será la siguiente:

```
void setup()
 {
    Serial.begin(115200);
```

Ha empezado iniciando e indicando la velocidad de la comunicación serial para monitorizar si todos los elementos iniciales fueron activados con éxito.

```
    touchAttachInterrupt(TOUCH_PIN_1, [](){}, touchThreshold);
    touchAttachInterrupt(TOUCH_PIN_2, [](){}, touchThreshold);
    touchAttachInterrupt(TOUCH_PIN_MAN, [](){}, touchThreshold);
    touchAttachInterrupt(touchNav_Pin, [](){}, touchThreshold);
```

Pasa a definir la función de los Pines táctiles, los cuales serán asociados a ser activados cuando se cumpla el umbral ya indicado las variables globales y también a que pin fue designado. La función lambda *([](){}),* significa que no se va a ejecutar ninguna función específica cuando se detecte un toque en estos pines.

```
    Wire.begin(LCD_SDA, LCD_SCL);
    lcd.init();
    lcd.backlight();
    lcd.clear();
    lcd.print("Iniciando...");
```

Estas líneas se encargaran de iniciar la pantalla LCD, comenzando con Wire.begin(...), la cual se encargará de asignar el bus I2C a los pines ya designados anteriormente, partiendo luego a comenzar los parámetros inició (*lcd.init*), encender la luz del fondo (*lcd.backlight*) y borrar cualquier carácter que esté presente (*lcd.clear*). Además de mostrar si fue activada con éxito.

```
    if (!radio.begin())
      {
        Serial.println("Error: Módulo RF24 no detectado!");
        while (1);
      }

    // Configurar radio
    radio.setChannel(108);
    radio.setDataRate(RF24_250KBPS);
    radio.setRetries(5, 15);
```

```
    radio.setPALevel(RF24_PA_LOW);
    radio.openReadingPipe(1, thisSlaveAddress);
    radio.openWritingPipe(masterAddress);
    radio.startListening();
```

Se seguirá la misma configuración que ya fue descrita en el código anterior, con el fin de permitir el uso del módulo nRF24L01+.

```
    initWiFi();
    server.on("/", handleRoot);
    server.on("/setMode", handleSetMode);
    server.on("/getData", handleGetData);
    server.begin();
    Serial.println("Sistema iniciado");
    Serial.print("Entrar en la Web con IP");
    Serial.println(WiFi.localIP());
```

- *initWiFi():* Esta función se encarga de establecer la conexión del ESP32 a la red WiFi utilizando las credenciales definidas en las variables globales.
- *server.on("/", handleRoot)*: se utiliza el objeto *server* (de la clase *WebServer*) para definir un "manejador" (handler) para la ruta raíz ("/") del servidor web. Cuando un cliente accede a la dirección IP del ESP32 sin especificar ninguna ruta adicional.
- *server.on("/setMode", handleSetMode):* Similar a la línea anterior, esta línea define un manejador para la ruta /setMode. Cuando un cliente accede a esta ruta, la función será ejecutada.
- *server.on("/getData", handleGetData):* Similar a la anterior, esta línea define un manejador para la ruta /getData. Cuando un cliente accede a esta ruta, la función será ejecutada.
- *server.begin():* Se encarga de iniciar el servidor web. Después de definir las rutas y sus correspondientes manejadores, esta función pone al servidor en modo de escucha, esperando las peticiones HTTP entrantes de los clientes en el puerto ya especificado (80).
- *Serial.println("Sistema iniciado"), Serial.print("Entrar en la Web con IP") y Serial.println(WiFi.localIP()):* Se encargaran de mostrar en el monitor serial que ha sido activada la web y la IP con la que se podrá ingresar.

LOOP

```
void loop()
 {
    server.handleClient();
    checkTouchPins();
    ActualizarUmbral();
    enviarUmbral();
    getData();
```

```
    botonNavegador();
    ActualizacionLCD();
    delay(10);
 }
```

El loop empieza llamando a la función *handleClient(),* fundamental para el funcionamiento del servidor web y descrita anteriormente.
Luego pasa a ejecutar las distintas funciones que ya serán activadas según el orden que fueron puestas en este loop. Además termina con una pequeña pausa de 10 milisegundos para reducir la velocidad del bucle y evitar que el microcontrolador consuma el 100% e su potencia de procesamiento, permitiendo que otras tareas se ejecuten de manera más eficiente.

Función initWiFi

```
void initWiFi()
  {
    Serial.println();
    Serial.print("Conectando a ");
    Serial.println(ssid);
```

Como se ha visto durante el inicio del código, empieza por esta función, la cual se encarga del proceso de intentar establecer una conexión a una red WiFi utilizando las credenciales proporcionadas.
Comenzando con los *Serial.print,* para mostrar en el monitor seria a la red que está conectado.

```
WiFi.begin(ssid, password);
```

Esta línea se encarga de iniciar el proceso de conexión, el cual intentará conectar el ESP32 a la red WIFI según las credenciales establecidas.

```
    unsigned long startAttemptTime = millis();
    while (WiFi.status() != WL_CONNECTED && millis() - startAttemptTime
< 10000)
    {
     delay(500);
     Serial.print(".");
    }
```

- *unsigned long start…: Se encarga de declarar la variable startAttemptTime,* asignando el valor la función *millis().*
- *While (WiFi.status()...):* Continuar ejecutando el código dentro del bucle MIENTRAS el ESP32 NO esté conectado a la red WiFi Y el tiempo transcurrido desde que se inició el intento de conexión sea menor de 10 segundos.
- Dentro de este bucle se crea una pausa de medio segundo *delay(500)* a la espera de volver a verificar el estado de la conexión WIFI. A su vez, también envia un punto al monitor serial para saber su estado.

```
    if (WiFi.status() == WL_CONNECTED)
```

```
    {
      Serial.println("\nWiFi conectado");
      Serial.print("IP: ");
      Serial.println(WiFi.localIP());
    } else {
     Serial.println("\nError WiFi");
    }
  }
```

- if (WiFi.status()...): Esta condición verifica el estado actual de la conexión WIFI, si el estado actual es igual a *WL_CONNECTED*, significa que la conexión a la red WiFi fue exitosa y pasará a mostrar en el Monitor Serial que ha sido conectado.
- En caso de que no se ha podido completar la conexión, se activa la condición *else*, mostrando que ha habido un error de conexión. Cerrando toda esta Función.

Función checkTouchPins

```
void checkTouchPins() {
   if (millis() - lastTouchTime < touchDebounce) return;
    if (touchRead(TOUCH_PIN_1) < touchThreshold) {
        manualMode = false;
        umbral = 30;
        lastTouchTime = millis();
        lcd.setCursor(0,0);
        lcd.print("Suelo poco humedo");
    }
```

Esta función se encarga de leer el estado de los diferentes pines táctiles y designar los distintos umbrales de humedad para activar el riego en la placa del Arduino. Por lo que empieza con:

- if (millis()...): Calcula la diferencia de tiempo en milisegundos entre el momento actual y la última vez que se detectó un toque. Si la diferencia de tiempo es menor que touchDebounce, significa que ha pasado muy poco tiempo desde el último toque detectado. En este caso, la función se detiene inmediatamente sin realizar ninguna otra acción.
- *if (touchRead(TOUCH_PIN_1) < touchThreshold) {...}:* Lee el valor actual del botón táctil asignado, y si cumple con el umbral de sensibilidad y se activara de la siguiente forma:
    - *manualMode = false:* Se encargará de desactivar el modo manual de umbrales si esta activo.
    - *umbral = 30:* asignará el nuevo umbral a esta variable.
    - *lastTouchTime = millis():* actualiza esta variable para el control anti-rebote.

- *lcd…:* Mostrará un mensaje informativo en la pantalla LCD indicando el nuevo modo.

```
    else if (touchRead(TOUCH_PIN_2) < touchThreshold) {
        manualMode = false;
        umbral = 70;
        lastTouchTime = millis();

        lcd.setCursor(0,0);
        lcd.print("Suelo casi humedo");
    }
    else if (touchRead(TOUCH_PIN_MAN) < touchThreshold) {
        manualMode = true;
        lastTouchTime = millis();
        lcd.setCursor(0,0);
        lcd.print("Modo manual activado");
    }
}
```

Los siguientes bloques, funcionarán de igual manera que el anterior descrito. Contando que la única acción relevante, es que cambiara la variable booleana *manualMode* a *true*, si es seleccionado su botón correspondiente.

Función ActualizarUmbral

```
void ActualizarUmbral() {
    if (manualMode) {
        int potValue = analogRead(thresholdPin);
        umbral = map(potValue, 0, 4095, 0, 100);
    }
}
```

Esta funcion se encargará de ajustar el valor de la variable global *umbral* basándose si está activado de este modo. por lo cual:

- if (manualMode) { ... }: Si la variable booleana es *true* se ejecutará. Declarando la variable *potValue*, encargada de administrar los valores analogicos del potenciómetro que está asignado al *thresholdPin.*
- *umbral = map(...):* los valores almacenados en la variable *potValue* seran convertidos mediante un remapeo de rango a valores a un rango de porcentaje de 0 a 100.

Función ActualizacionLCD

```
void ActualizacionLCD() {
    static unsigned long lastUpdate = 0;
    if(millis() - lastUpdate < 500) return;
    lcd.setCursor(0, 0);
```

Se ha dispuesto esta función para controlar y actualizar la información que se muestre en la pantalla LCD.

- static unsigned long…: esta variable se inicializa sólo la primera vez que se llama a la función *ActualizacionLCD()* y su valor se conserva entre las diferentes llamadas a la función. por lo cual se utilizará para controlar la frecuencia de actualización de la pantalla LCD, implementando un retardo para evitar actualizaciones demasiado rápidas que podrían causar parpadeo o un uso innecesario de recursos.
- if(millis()...): Si la diferencia de tiempo es menor a 500 ms, se detiene sin realizar ninguna otra acción.
- lcd.set…: se ha puesto fuera de los programas descritos a continuación, ya que será usado de la misma manera, evitando aumentar las líneas de código.

```
switch(lcdScreen) {
        case 0: // Pantalla principal
        {
            lcd.clear();
            lcd.print("Humedad 1: ");
            lcd.print(newData.humedadSuelo1);
            lcd.print("%");
            lcd.setCursor(0,1);
            lcd.print("Humedad 2: ");
            lcd.print(newData.humedadSuelo2);
            lcd.print("%");

            break;
        }
        case 1:
        {
            lcd.clear();
            lcd.print("Humedad Aire:");
            lcd.print(newData.humedadAire, 0);
            lcd.print("%");
            lcd.setCursor(0, 1);
            lcd.print("Temperatura: ");
            lcd.print(newData.temperatura, 1);
            lcd.print((char)223);
            lcd.print("°C");
            break;
        }
        case 2: // Pantalla de configuración
         {
            lcd.clear();
            lcd.print("Umbral en:");
```

```
            lcd.print(umbral);
            lcd.print("%");
          lcd.setCursor(0, 1);
          lcd.print("Programa: ");
              lcd.print(umbral == 30 ? "1" : umbral == 70 ? "2" :
"Manual");
          lcd.print("  ");
          break;
        }
      case 3: // Pantalla de conexión
        {
          lcd.clear();
          lcd.print("La Web con IP");
          lcd.setCursor(0, 1);
          lcd.print(WiFi.localIP());
        }
    }
      lastUpdate = millis();
}
```

- switch(lcdScreen) {...}: Inicia una estructura de control *switch.* La expresión dentro de los paréntesis *(lcdScreen*) se evalúa, y el flujo del programa se dirige al case cuya etiqueta coincida con el valor de esta expresión.
- *case 0, 1, 2 o 3{}*: se interpreta que solo ha sido presionado el touch Pin el número correspondiente de veces, por lo cual ejecutará las funciones que estén dentro de cada case.
- En el case 0 y 1, se encargaran de mostrar en la pantalla LCD, mediante sus funciones los distintos sensores que han sido recibidos por el modulo nRF24L01+.
- Para el case 2, visualiza que programa se ha seleccionado y el umbral manual que esté siempre puesto.
- Terminando con el case 3, solamente visualizará la dirección IP por la cual se podrá acceder a la interfaz Web.
- Todos contarán con un *break;* al final para indicar que tendrá que salir de ese case y se podrá ejecutar el siguiente, según se desee.

Función botonNavegador

```
void botonNavegador()
 {
    const int touchNav = touchRead(touchNav_Pin);

    if(touchNav < touchThreshold) { // Botón presionado
        static unsigned long lastTouchTime = 0;
        if (millis() - lastTouchTime > 900) {
```

```
        lcdScreen = (lcdScreen + 1) % 4; // Rotar entre 0, 1, 2
        lcd.clear();
        lastTouchTime = millis();
    }
    }
}
```

Como se ha visto en las lineas anteriores, hará necesario de la variable *lcdScreen* para la ejecución de los case, por lo cual, esta función se encarga de detectar pulsaciones en el pin táctil dedicado a la navegación por las pantallas de la LCD y cambiar la pantalla mostrada en la LCD en respuesta a esas pulsaciones.

- *const int touchNav:* se declara esta variable, en función de si ha sido presionado el touch Pin asignado para este trabajo.
- *if (touchNav < touchThreshold) {...}:* Compara el valor leído en la anterior variable y si cumple con el umbral de sensibilidad, ejecutará el siguiente bloque:
    - *static unsigned long last…:* Esta variable conserva su valor entre las llamadas a la función, y se utiliza para implementar un mecanismo anti-rebote para el botón de navegación.
    - *if (millis()...):* calcula la diferencia de tiempo en milisegundos entre el momento actual y la última vez que se detectó una pulsación válida del botón de navegación. Si ha transcurrido suficiente tiempo desde la última pulsación válida, se ejecutará el siguiente bloque:
    - *lcdScreen = (lcdScreen + 1) % 4:* Incrementa el valor de la variable global *lcdScreen* en 1 y luego aplica el operador módulo (% 4). Esto hace que *lcdScreen* rote cíclicamente entre los valores 0, 1, 2 y 3. Usados en la función anterior.
    - *lastTouchTime = millis():* Actualiza la variable con el tiempo actual, registrando el momento de esta pulsación válida para el control anti-rebote.

Función enviarUmbral

```
void enviarUmbral() {
    static unsigned long lastSend = 0;
    static int lastUmbral = -1;

    if (umbral != lastUmbral || millis() - lastSend > 500) {
        radio.stopListening();
        delay(10);
        snprintf(dataToSend, sizeof(dataToSend), "u:%d", umbral);
```

Trabajando similar a la función *enviarDatosSensores,* ésta se encargará de enviar los valores umbrales a la *estación de activación,* por lo cual empieza de la forma:

- *static unsigned long lastSend…*: Se declara una variable estática, conservando su valor entre las llamadas a la función y se utiliza para almacenar la marca de tiempo.
- *static int lastUmbral…:* Se declara esta variable local con este valor para indicar cuando un valor umbral no será válido.
- *if (umbral != lastUmbral || millis() - lastSend > 500) { ... }:* Esta condición se ejecutará si el valor actual del umbral es diferente del último valor enviado O si han pasado más de 500 milisegundos desde el último envío.
- *radio.stopListening:* cuando la condición antes descrita es activada, esta línea se encargará de desactivar la lectura del módulo nRF24L01+. Contando con un delay a continuación para estabilizarlo.
- *snprintf(dataToSend…):* esta linea se encargará de transmitir la cadena que tendrá el formato "*u:XX*" donde "XX" será el valor actual del umbral de humedad y el prefijo "u:" ser utilizado por la estación de activación para identificar el tipo de dato que se está recibiendo.

```
        bool success = radio.write(&dataToSend, sizeof(dataToSend));
        if (success) {
            Serial.print("Umbral enviado: ");
            Serial.print(umbral);
            Serial.println(manualMode ? "% (Manual)" : "% (Preset)");
            lastUmbral = umbral;
            lastSend = millis();
```

- *bool success…:* Justo después de crear el formato del dato a enviar, esta función boleana se encarga de intentar enviar los datos a través del módulo nRF24L01+, especificando la ubicación donde comienza la cadena de texto, el tamaño total del buffer. Si la transmisión ha sido exitosa, se podrá en *true,* permitiendo la activación del siguiente bloque.
- *if (success) { ... }:* Si la transmisión descrita anteriormente ha sido exitosa, se encarga de activar este bloque. Donde indicará en el Monitor Serial que datos fueron enviados.
- *lastUmbral = umbral:* Actualiza cual fue el último umbral enviado.
- *lastSend = millis():* Almacena la marca del tiempo del ultimo envio exitoso.

```
        } else {
            Serial.println("Error al enviar");
        }
        radio.startListening();
    }
}
```

En el caso de que no hubo transmision exitosa, se encargara de notificar en el Monitor Serial que hubo un error al enviar.
Esta función se cierra permitiendo de nuevo la lectura de datos recibidos.

Función getData

```
void getData() {
    if (radio.available()) {
        radio.read(&newData, sizeof(newData));
        Serial.print("Datos recibidos - S1:");
        Serial.print(newData.humedadSuelo1);
        Serial.print("%, S2:");
        Serial.print(newData.humedadSuelo2);
        Serial.print("%, Aire:");
        Serial.print(newData.humedadAire);
        Serial.print("%, Temp:");
        Serial.print(newData.temperatura);
        Serial.println("°C");
    }
}
```

Esta funcion se encarga de verificar si hay datos disponibles para ser recibidos a través del módulo nRF24L01+.

- if (radio.available()) { ... }: Esta línea verifica si hay datos disponibles para ser leídos en el buffer de recepción del módulo de radio. En el caso que si existan datos, se podrá en *true* y se activará este bloque, y *false* si no cuenta con nuevos datos.
- radio.read(&newData…): Esta función lee los datos disponibles del módulo de radio y los almacena en la memoria. Proporcionando la dirección de memoria u especificando la cantidad de bytes que espera recibir
- *Serial.print():* Las siguientes líneas con esta función, mostrará los datos de los sensores de la *estación de activación* suministrados.

Función getWebPage

Debido a la gran cantidad de líneas de código que contiene esta función, utilizadas en su mayoría para el funcionamiento gráfico de la interfaz Web, se describirán solamente las más relevantes. El resto de todas sus líneas están disponibles en el anexo 13 de todo el código del ESP32.

```
    String page = "<!DOCTYPE html><html><head><title>McGarden Labs</title>";
```

Esta línea inicia la construcción de la página HTML.

```
    page += "<meta http-equiv='refresh' content='2'>";
```

Se estabelece la auto-actualización de la página cada 2 segundos.

```
    page += "<link rel='stylesheet' href='https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.min.css'>";
```

Se refiere a la inclusión de una hoja de estilos externa desde una red de entrega de contenido (CDN) llamada cdnjs para utilizar la librería de iconos Font Awesome.

```
    page += "<h2><i class='fas fa-tint'></i> Humedad Suelo 1</h2>";
          page  +=  "<div  id='soil1-value'  class='sensor-value'>"  +
String(newData.humedadSuelo1) + "%</div>";
    page += "</div>";

    page += "<div class='sensor-card'>";
    page += "<h2><i class='fas fa-tint'></i> Humedad Suelo 2</h2>";
   page   +=   "<div   id='soil2-value'   class='sensor-value'>"   +
String(newData.humedadSuelo2) + "%</div>";
    page += "</div>";

    page += "<div class='sensor-card'>";
    page += "<h2><i class='fas fa-cloud'></i> Humedad Aire</h2>";
          page  +=  "<div  id='air-value'  class='sensor-value'>"  +
String(newData.humedadAire, 0) + "%</div>";
    page += "</div>";

    page += "<div class='sensor-card'>";
           page   +=   "<h2><i   class='fas   fa-thermometer-half'></i>
Temperatura</h2>";
          page  +=  "<div  id='temp-value'  class='sensor-value'>"  +
String(newData.temperatura, 1) + "C</div>";
```

La realización de estas líneas establece la visualización de los valores de los sensores que han sido transmitidos entre los módulos nRF24L01+, siguiendo la estructuras ya mencionadas.

```
    page += "<div class='progress-container'>";
           page   +=   "<div   class='progress-label'><span>Umbral   de
humedad:</span>   <span   id='umbral-value'>"   +   String(umbral)   +
"%</span></div>";
          page  +=  "<div  class='progress'><div  id='progress-bar'
class='progress-bar'  style='width:"  +  String(umbral)  +  "%'>"  +
String(umbral) + "%</div></div>";
```

Estas líneas han sido puestas para crear la línea de progreso que funcionará en base a los valores del umbral.

```
    page += "<h2><i class='fas fa-cogs'></i> Modo de Operacion</h2>";
    page += "<div class='btn-group'>";
    page += "<button id='preset30' class='btn btn-preset";
    if (umbral == 30 && !manualMode) page += " active";
        page   +=   "'   onclick='setMode(\"preset30\")'><i   class='fas
fa-leaf'></i> Suelo poco humedo</button>";
```

```
    page += "<button id='preset70' class='btn btn-preset";
    if (umbral == 70 && !manualMode) page += " active";
        page   +=   "'   onclick='setMode(\"preset70\")'><i   class='fas
fa-tree'></i> suelo casi humedo</button>";

    page += "<button id='manual' class='btn btn-manual";
    if (manualMode) page += " active";
    page    +=    "'    onclick='setMode(\"manual\")'><i    class='fas
fa-sliders-h'></i> Manual</button>";
```

Este segmento está asignado a los botones, el cual comenzará con el texto que indicará los distintos modos de operación. Ha sido elaborado para que al presionar cada caja de texto, se active el umbral preestablecido.

```
    page += "<script>";
    page += "function setMode(mode) {";
       page  +=  "    fetch('/setMode?mode='  +  mode).then(response  =>
response.text()).then(data => {";
    page += "    console.log('Mode set:', data);";
    page += "    updateStatus();";
```

Este bloque será tratado como código JavaScript, por lo cual comienza abriendo con una etiqueta *<script>* .

- function setMode(mode) { ... }: establece la funcion setMode, la cual se encargara mediante la linea fetch('/setMode?mode=' + mode), solicitar una petición HTTP GET al servidor web del ESP32 a la ruta */setMode,* enviando el valor del *mode* como un parámetro en la URL
- *.then(response => response.text()):* Seguidamente, cuando la peticion al servidor es exitosa, esta parte recibe la respuesta y la formatea como texto, ejecutando las siguientes líneas.
- *console.log('Mode set:', data):* Imprime un mensaje en la consola del navegador indicando que el modo se ha establecido y mostrando la respuesta del servidor.

```
    page += "function updateStatus() {";
            page    +=    "       fetch('/getData').then(response    =>
response.text()).then(data => {";
    page += "    const parts = data.split(',');";
      page  +=  "     document.getElementById('soil1-value').innerText  =
parts[0] + '%';";
      page  +=  "     document.getElementById('soil2-value').innerText  =
parts[1] + '%';";
       page  +=  "      document.getElementById('air-value').innerText  =
parts[2] + '%';";
      page  +=  "     document.getElementById('temp-value').innerText  =
parts[3] + '°C';";
```

```
    page += "  });";
    page += "}";
```

Siguiendo la última línea del bloque anterior, se declara function updateStatus() {, para obtener los datos de los sensores del ESP32 a través de una petición web y actualizar los valores correspondientes mostrados en la página web.

- *fetch('/getData')....:* Inicia una petición a la función *handleGetData() (explicada más adelante),* configurada para responder a la transmisión de los últimos datos de los sensores.
- const parts = data.split(','): Divide la cadena de texto data en un array de subcadenas llamado parts. El método *split(',')* utiliza la coma (,) como delimitador para separar los diferentes valores de los sensores.
- document.getElementById…: las siguientes líneas que inician de esta manera, se encargan de buscar en el Document Object Model (DOM) de la página web el elemento HTML que tiene el atributo *id* con el valor especificado.

```
    page += "setInterval(updateStatus, 2000);";
    page += "</script>";

    page += "</div></body></html>";
```

- setInterval(...): Designa que será cada dos segundos cuando deberá ser actualizada esta función en el JavaScript.
- *"</script>":* Cierra la lectura del código en JavaScript.

Función hadleRoot

```
void handleRoot() {
    server.send(200, "text/html", getWebPage());
}
```

Esta función se encarga de responder a las peticiones HTTP que se realizan a la raíz ("/") del servidor web del ESP32. Cuando el cliente accede a la dirección IP del ESP32 sin especificar ninguna ruta adicional.

Función hadleSetMode

```
    if (server.hasArg("mode")) {
        String mode = server.arg("mode");

        if (mode == "preset30") {
            manualMode = false;
            umbral = 30;
            server.send(200, "text/plain", "Preset 30%");
        } else if (mode == "preset70") {
            manualMode = false;
            umbral = 70;
```

```
            server.send(200, "text/plain", "Preset 70%");
        } else if (mode == "manual") {
            manualMode = true;
            server.send(200, "text/plain", "Manual");
        } else {
            server.send(400, "text/plain", "Modo inválido");
        }
    } else {
        server.send(400, "text/plain", "Falta parámetro mode");
    }
}
```

Esta función se encarga de recibir y procesar las peticiones *HTTP GET* que se realizan a la ruta */setMode* del servidor.

- *if (server.hasArg("mode")) { …: Este* bloque será activado si la petición suministrada por el servidor contiene un argumento *"mode",* dentro de la URL y trabaja con las siguiente funciones.
- *String mode =...:* Se encargará de almacenar el modo que ha sido establecido dentro de la interfaz web.
- *if (mode == "manual") { ... }, else if (mode == "preset…") { ...):* Según los valores que contenga la variable *mode,* se activará uno de estos bloques. Enviando el umbral a la Web con la línea *server.send(200, "text/plain"...*
- *else { y server.send…:* Cuando este parámetro *mode*, no cuente con ningún valor válido, se encargará de notificar que hubo un error, mediante el código 400.

Función hadleGetData

```
void handleGetData() {
    String data = String(newData.humedadSuelo1) + "," +
                  String(newData.humedadSuelo2) + "," +
                  String(newData.humedadAire, 0) + "," +
                  String(newData.temperatura, 1);
    server.send(200, "text/plain", data);
}
```

Esta función se encarga de responder a las peticiones HTTP GET que se realizan a la ruta */getData* del servidor web del ESP32. Proporcionando los datos actuales de los sensores.

- String data = String(...) + *"," +:* Estas líneas construyen cadenas de textos provenientes de los datos suministrados por los módulos nRF24L01+.

5. Planificación

El desarrollo del proyecto fue realizado en un total de 100 horas, distribuidas a lo largo de 4 meses desde su planteamiento, hasta la entrega final de esta memoria. Por lo cual, se consideró desarrollar en distintas etapas su elaboración, como se ve en la Figura 68.

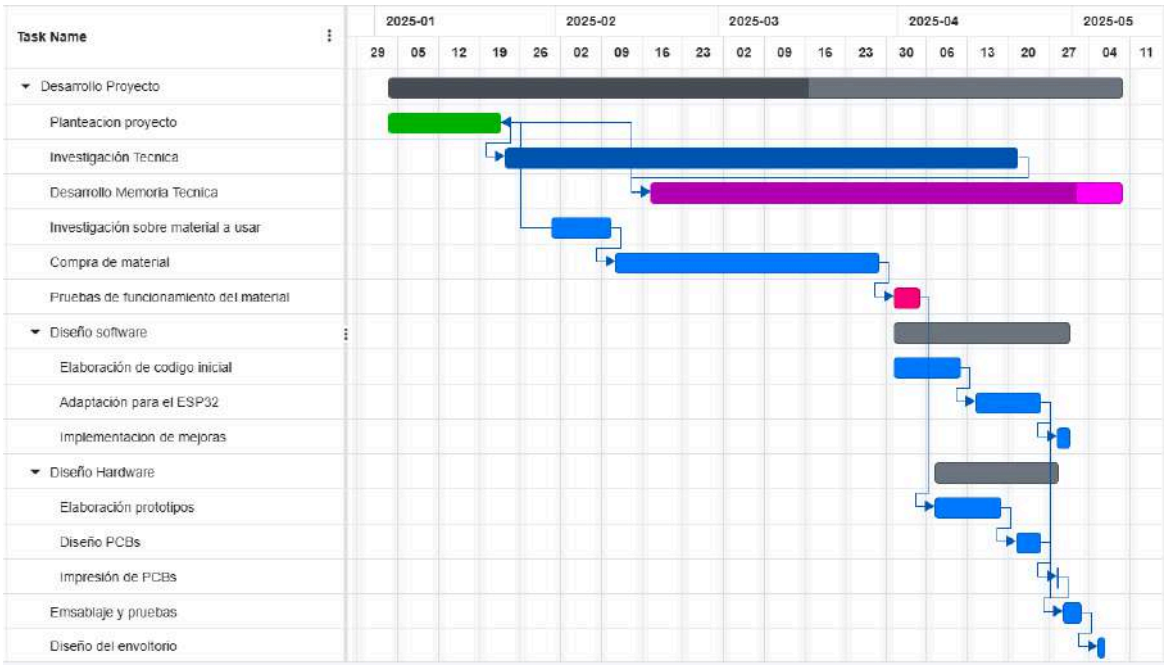


Figura 74: Diagrama de Gantt.
Fuente: Elaboración propia.

Para la planificación fue puesto a consideración las siguientes tareas:

- *Planeación del proyecto*: Desde el inicio del curso, el profesor comentó que se podía comenzar a realizar la investigación necesaria para la elaboración de los proyectos, por lo cual se decidió el planteamiento de este desde la primera semana de enero. Esta tarea consistió en el análisis de proyectos elaborados anteriormente por otros estudiantes del instituto, como también podrían ser realizados por estudiantes de otras instituciones. Este análisis inicial sirvió para entender diferentes enfoques y tener una visión más amplia de las posibilidades y los retos que se encontrarán.

- *Investigación Técnica*: Para la realización del proyecto se requirió desde el inicio realizar una investigación técnica sobre el funcionamiento de los diversos dispositivos que serían usados y demás aspectos, por lo cual esta tarea se desarrolló durante todo el tiempo que fue necesario. Esta investigación fue fundamental para comprender a fondo las características y el modo de operación de cada componente, desde los sensores hasta los microcontroladores y los módulos de comunicación inalámbrica.

- *Desarrollo de memoria*: Con el fin de registrar todo el proceso y cumpliendo con el requerimiento de la presentación del proyecto, el desarrollo del presente documento empezó a ser redactado en el momento de la elección de que se realizaría. La idea fue ir documentando los avances de forma continua para no perder detalles importantes del desarrollo.

- *Elección y compra de material*: Después de realizar una primera aproximación de qué sería necesario para la realización del proyecto, se eligieron los materiales más idóneos en cuanto a sus prestaciones, coste y disponibilidad. Se solicitó su compra al profesor, especificando las características técnicas relevantes de cada componente.

- *Pruebas*: Para conocer los aspectos de funcionamiento de los materiales usados que formarían parte del proyecto, se decidió realizar pruebas que determinarían si contaban con un óptimo funcionamiento. Estas pruebas iniciales permitieron verificar que cada componente funcionaba como se esperaba y obtener datos preliminares sobre su comportamiento.

- *Diseño de Software:* Determinando los aspectos digitales que abordaría el proyecto, se comenzó a desarrollar esta tarea a la par que se tuvieron los materiales, para ir desarrollando las líneas de código según cómo trabajaría con los componentes. Además, al poder ser modificado para intentar obtener un mejor resultado del que ya tendría, se planteó que tuviera una fase para posibles mejoras y optimizaciones del código.

- *Diseño de Hardware:* Una vez se ha comprobado si todos los materiales funcionaran y conociendo sus características después de realizar las pruebas, se comenzó con la elaboración de los prototipos, para proceder a la elaboración de las PCBs, siguiendo los requerimientos que fueron establecidos a la hora de su realización.

- *Ensamblaje y pruebas:* Después de contar ya con el diseño del Software y hardware, el proyecto pasó a sus pasos finales, los cuales fueron de ensamblar todos sus componentes en función de cómo trabajan y se realizaron diversas pruebas para monitorear que cumplieran con los estándares ya planteados. Estas pruebas finales buscaron asegurar la correcta integración de todas las partes del sistema y verificar su funcionamiento global.

- *Diseño Envoltorio:* Para finalizar todos los aspectos tanto físicos como digitales, se diseñó el paquete donde sería presentado, cumpliendo los requisitos deseados y siendo llamativo para los espectadores a los que sería presentado.

Pese a haber sido establecido objetivos a conseguir en cada apartado, el proyecto contó con notables retrasos, producidos por ciertos problemas que frenaron continuamente su desarrollo, como fueron:

- Demora en entrega de material por parte del proveedor, lo que retrasó la fase de pruebas iniciales del hardware.
- Adaptación del código del NodeMCU al ESP32, requiriendo tiempo adicional de investigación y depuración.
- Inestabilidad en el módulo nRF24L01+ por falta de un condensador de desacoplo, lo que generó fallos intermitentes en la comunicación inalámbrica hasta su identificación y solución.
- Falta de conocimiento de herramientas de diseño 3D para la elaboración más profesional del envoltorio, limitando la calidad del diseño final.
- Falta de establecimiento claro de fechas de entregas de memoria, lo que provocó una gestión del tiempo menos eficiente en la documentación.
- El intento de crear funcionalidades extras a la interfaz Web, que finalmente se descartaron por limitaciones de tiempo.

6. Costes

Como se ha visto reflejado en la planificación, el desarrollo del proyecto fue realizado en distintas etapas, de las cuales, solo será considerado para el presente análisis, las fases de la elaboración del software y diseño de Hardware.

Mano de obra
La determinación del costo de horas totales dedicadas en la elaboración del proyecto, fue realizada siguiendo las tablas salariales del convenio colectivo del sector siderometalúrgico de barcelona 2024, donde podemos concluir que:

- *Desarrollo de software:* Grupo 3; programador, por la codificación de programas de ordenador. Con un costo de ***15,26*** la hora.
- *Diseños de PCB:* Grupo 2, Ingeniero, debido a que su elaboración fue realizada con requisitos ya establecidos. Su costo será de ***16,54*** la hora.
- *Ensamblaje y Pruebas:* Grupo 3; Operario, se encarga de realizar la soldadura y verificación del buen funcionamiento de los materiales que la componen. Su costo será de ***14,98*** la hora.

Según la información dada, se ha determinado que al contar con 25 horas del desarrollo del código, 15 horas del diseño de PCB y 6 de ensamblaje y pruebas técnicas. El costo total fue de ***719.48€.***

Herramientas de desarrollo
Como se ha mencionado a lo largo del proyecto, se utilizó programas como easyEDA y Arduino IDE junto a las librerías de los componentes que son de acceso público, las cuales no representan ningún gasto por parte de sus licencias, ya que son herramientas openSource.

Fabricación de PCBs y componentes
El coste de fabricación no ha sido asumido, ya que fue elaborado con las herramientas propias del centro, sin embargo, si hubiera sido fabricada de forma externa, su costo estimado sería de *10€*. Por lo cual, solo se asume el precio de la *Placa de fibra de vidrio fotosensible positiva de 100 x 160 mm,* la que tiene un costo de **6,58*€.*** Además de contar con los siguientes componentes que fueron usados para su ensamblaje:

- 4 Tira de Conectores de Pin Hembra : ***1,39€.***(1)
- 3 Tira de Conectores de Pin Macho: ***1,23€.***(1)
- 1 Tira de Conectores de Pin Macho en L: ***0,99€.***(1)
- 2 Condensadores Electrolíticos de 10 UF: ***0,20€.*** (2)
- 1 Potenciometro de 10k: ***0,85€.*** (2)
- 1 Tira de cables dupont mixtos: ***2,99€***(1)

Dando un resultado total de ***18,07€*** en el costo de los materiales elaborados para su fabricación.

Componentes externos

El coste que tuvieron todos demás componentes usados en el proyecto fueron:

- 1 Arduino Uno: ***1,29€.*** (1)***.***
- 1 ESP32: **9,50*€.*** (2)**.**
- 1 Modulo IRF520: ***3,99€.*** (2)***.***
- 2 Modulo nRF24L01+: ***3,25€.*** (2).
- 2 Sensor de humedad para tierra con módulo LM393: ***5,45€.*** (2)***.***
- 1 Sensor de Temperatura y Humedad DHT11: ***7,56€.*** (2)***.***
- 1 Pantalla LCD con módulo I2C: ***9,26€.*** (2)***.***
- 1 Bomba de agua sumergible 5V: **9,50*€.*** (2)**.**

Con un coste total de ***58.5€***

Consideraciones

El coste de los materiales comentados, cuentan con el IVA aplicado de 21%.

Se ha descartado el valor del envoltorio, ya que fue realizado con materiales reutilizados, como el caso del cartón y las chapas metálicas.

No ha sido registrado exactamente el tiempo dedicado a la memoria técnica, debido a que fue desarrollado a lo largo de toda la elaboración del proyecto, por lo cual no podrá contar con un valor determinado.

(1) Compras realizadas en Aliexpress, (2) Compras realizadas en diotronic.

Concepto	**Unidad**	**Coste**	**Total**
Mano de obra			
Desarrollo del software	25 Horas	15,26	381,50
Diseño de PCBs	15 Horas	16,54	248,10
Ensamblaje y Pruebas	6	14,98	89,88
Fabricación PCBs			
Placa de fibra de vidrio fotosensible positiva de 100 x 160 mm	1	6,58	6,58
Tira de Conectores de Pin Hembra	4	1,39	5,56
Tira de Conectores de Pin Macho	3	1,23	3,69
Tira de Conectores de Pin Macho en L	1	0,99	0,99
Condensador Electrolítico de 10UF	2	0,20	0,40
Potenciometro de 10k	1	0,85	0,85
Tira de cables dupont mixtos	1	2,99	2,99

Componentes Externos			
Arduino Uno	1	1,29	1,29
ESP32	1	9,50	9,50
Modulo IRF520	1	3,99	3,99
Modulo nRF24L01+	2	3,25	3,25
Sensor de humedad para tierra con módulo LM393	2	5,45	10,90
Sensor de Temperatura y Humedad DHT11	1	7,56	7,56
Pantalla LCD con módulo I2C	1	9,26	9,26
Bomba de agua sumergible 5V	1	9,50	9,50
Total			***795.79***

Tabla 9: Coste total.
Fuente: Elaboración propia.

Conclusiones

Como se puede observar, el coste total del proyecto recae principalmente en la mano de obra necesaria para su creación, por lo cual, podrá ser amortiguado en el caso de ser fabricado en serie. Además, como se ha mencionado en el apartado 5.2, el coste de los materiales utilizados para su funcionamiento podría ser más reducido al realizar su compra con otros distribuidores.

La realización del proyecto cuenta con la posibilidad de ser mejorado en un futuro, tal como se ha podido comprobar, por lo cual, un posible desarrollo con fines comerciales podría ser la integración de los componentes que hacen parte de los módulos utilizados para el funcionamiento de este, lo que podría disminuir significativamente el costo de elaboración, ya que su fabricación en serie los hace mas económicos, por ejemplo: la producción de las PCBs que por unidad su valor es de 10€, pero al ser una cantidad mayor pasaría a ser de hasta 2€ o tambien el modulo IRF520, que su valor es de 3,99€, pero sus componentes necesarios para ejecutar su función, no pasaría de 2€, sin contar que a más cantidad, su valor disminuye.

7. Análisis de impacto Social y Ambiental

Actualmente, el desarrollo de sistemas de riego en diversos sectores a menudo depende de sistemas tradicionales que requieren una supervisión manual intensiva por parte del personal. Esta dependencia no solo conlleva un considerable esfuerzo físico y mental para los trabajadores, sino que también puede resultar ineficiente en términos de optimización del uso de recursos hídricos. Reconociendo esta necesidad de modernización y mejora de la calidad de vida de los operarios, este proyecto se ha orientado al desarrollo de mecanismos tecnológicos capaces de monitorizar diversos parámetros cruciales para el cuidado del suelo y la salud de las plantas de manera automatizada.

Además, se ha priorizado la creación de una interfaz de usuario intuitiva y accesible, minimizando la curva de aprendizaje y eliminando la necesidad de una formación tecnológica exhaustiva para su correcta utilización.

El proyecto no solo aspira a aliviar la carga de trabajo arduo asociada al riego manual, sino que también se enfoca en un objetivo ambiental de gran relevancia: la reducción del consumo ineficiente de agua. Ya que la falta de información precisa sobre las condiciones de humedad del suelo a menudo conduce a un riego excesivo o insuficiente, mediante la implementación de sensores inteligentes y un sistema de control adaptativo, se estima que este proyecto tenga el potencial de disminuir el consumo de agua no aprovechable entre un 20% y un 50%. Esta reducción no solo implica un ahorro económico significativo para los usuarios, sino que también contribuye a la conservación de un recurso natural cada vez más valioso y limitado.

Adicionalmente, se ha decidido que el proyecto siga una clara línea de eficiencia energética. Por lo que el diseño del hardware cuenta con componentes que demandan un bajo consumo eléctrico, esta característica puede traducirse en un ahorro sustancial en la demanda eléctrica en comparación con sistemas de riego inteligente convencionales, que a menudo incorporan elementos con un consumo energético más elevado. Al optimizar el uso de la energía, el proyecto busca minimizar su huella ambiental global, promoviendo una solución de riego que sea tanto inteligente como sostenible.

Como podemos observar, este proyecto de riego inteligente responde a la necesidad de sistemas más eficientes, sostenibles y que alivien la carga de trabajo manual. Al proporcionar información precisa, automatizar el control y optimizar el uso de recursos como el agua y la energía, se busca generar un impacto positivo tanto a nivel social, como ambiental.

8. Conclusiones

El proyecto ha logrado desarrollar un sistema de riego inteligente pudiendo cumplir en su mayoría con los objetivos establecidos a la hora de su desarrollo, proporcionando una solución eficiente y automatizada para el cuidado de las plantas. Basado en la combinación del entorno arduino y el ESP32, ha permitido realizar una monitorización remota de las condiciones de humedad del suelo y el control preciso del riego a través de una interfaz web y física intuitiva. Reduciendo el esfuerzo manual requerido para el mantenimiento de jardines y pequeños cultivos.
El diseño modular del sistema, dividido en una estación de activación y una estación de control, demuestra ser una solución eficaz para la gestión del riego en diferentes áreas, proporcionando las bases para futuras expansiones y mejoras, como podría ser: unos mejores sensores, la integración en una sola placa reduciría significativamente los costes de producción en serie, haciendo el sistema más accesible a un público más amplio, o integración con aplicaciones web que almacenan el historial de riego permitiría a los usuarios analizar tendencias, optimizar el uso del agua a largo plazo y detectar posibles problemas en el sistema.

Es cierto que, debido a la escasez de tiempo, cambios en el diseño y retraso en la entrega de materiales, no se ha profundizado extensamente en ciertas secciones del proyecto, dejando una especificación brevemente comentada.
Señalar también que el desarrollo ha supuesto una serie de evoluciones en el diseño y en la especificación, enriqueciendo directamente el resultado final. Creando así un producto competente en precio y funcionalidad dentro del mercado.
Como ya se ha mencionado en el apartado *Costos*, el proyecto supone mayormente una inversión en mano de obra. Dejando un coste de proyecto algo elevado pero amortizable
con el bajo coste de producción.

Pese a los tropiezos que se vio envuelto la realización del proyecto, he podido afianzar muchos conceptos claves que fueron enseñados durante las clases de electrónica digital y analogica, tambien de mantenimiento de equipos de radiofrecuencia, como también conocer en mayor medida, muchos aspectos claves a la hora de diseñar un proyecto que podría abrir las puertas al desarrollo de sistemas domóticos que faciliten tareas arduas.
Gracias al desarrollo del proyecto, he podido realizar distintas tareas que podrían entrar en el campo de la ingeniería, pero con limitaciones debido al tiempo que terminó siendo muy reducido.
Además de mencionar en más de una ocasión las capacidades que tiene el proyecto para posibles mejoras en cuanto a diseño. A lo largo de la finalización de la memoria he podido constatar el potencial que cuenta, para que sea un producto que cumpla con la normativa actual y poder ser comercializado con el fin de automatizar sistemas de riego.

Posibles mejoras

Aunque el proyecto ha sido diseñado siguiendo unos requisitos mínimos para ser considerado de sistema de riego inteligente y demostrando que ha podido cumplir con los objetivos propuestos, puede contar con un amplio margen de mejoras, para mejorar su rendimiento o funcionalidades, incluso aumentando las ramas de desarrollo relacionadas a su función.

Hardware:

- Optimizar las dimensiones del diseño, reduciendo su tamaño para facilitar su instalación en espacios reducidos o integrarlo de forma más discreta en el entorno.
- Rediseñar los módulos que componen cada estación. Permitiendo ser elaboradas sus características en uno.
- Diseñar un envoltorio más robusto y resistente a la intemperie, que proteja los componentes electrónicos de la humedad y el polvo, prolongando la vida útil del sistema en exteriores.
- Sustituir y seleccionar sensores de humedad del suelo con mayor resistencia a la corrosión y la degradación por el uso continuo en ambientes húmedos. Siendo útil para datos más precisos.
- Incluir conectores que ofrezcan una mayor seguridad de conexión, resistencia a vibraciones y a las entradas de agua y polvo.

Software:

- Reestructurar el código, facilitando la adición de nuevas funcionalidades o la adaptación del sistema a diferentes necesidades sin afectar a las partes existentes.
- Añadir gráficos interactivos para visualizar los datos de los sensores, implementar un sistema de alertas y notificaciones, o permitir la gestión de múltiples zonas de riego.
- Mejorar la usabilidad de la interfaz web mediante la inclusión de documentación integrada y textos de ayuda contextuales.
- Permitir la integración con plataformas web de almacenamiento de datos (como Firebase o ThingSpeak) para registrar el historial de riego, analizar tendencias y facilitar la gestión remota del sistema a través de internet.
- Implementar la comunicación mediante el protocolo MQTT para permitir la integración con otros sistemas de domótica y plataformas IoT.
- Diseñar algoritmos que mediante los datos recogidos, pueda aplicar soluciones autónomas que optimicen la cantidad de agua empleada y maximicen el crecimiento de la planta.
- Ampliar las opciones de control disponibles para el usuario a través de la interfaz web, proporcionando mayores botones que funciones para más eventos.
-

9. Webgrafia

1. Historia de la automatización, de Historia de la Automatización - LaHistoria.
2. Sistemas de riego IoT: Ahorra agua inteligente, de Sistemas de Riego IoT: Ahorra Agua Inteligente.
3. Kevin Ashton y el origen del Internet de las Cosas, de ¿Qué es el Internet de las Cosas (IoT)?
4. Comparación de sistemas de riego, de COMPARACION SISTEMAS DE RIEGO - Gestiriego.
5. Estrategias para el diseño de sistemas de riego inteligentes, de Estrategias para el diseño de sistemas de riego inteligentes | DigiKey.
6. *ESP32 vs ESP8266: Comparación de especificaciones técnicas*, de ESP32 vs ESP8266 ¿Cuales son las diferencias entre ambos módulos?
7. *Sensores de humedad del suelo en riego inteligente*, de Sensores de Humedad del Suelo | HIBA Formación.
8. Selección e instalación de válvulas solenoides para riego, de Selección e Instalación de Válvulas Solenoides para Riego | Tameson.es.
9. Convenio colectivo del sector siderometalúrgico de barcelona 2022-2024, de llibret-convenio-metal-barcelona-2022-2024.pdf

10. Apéndices

Anexo 1: Características Técnicas Sensor DHT11

Sensores y MCU

Parámetros	Condiciones	Mínimo	Típico	Máximo
Humedad				
Resolución		1%HR	1%HR	1%HR
			8 bits	
Repetibilidad			±1%HR	
Precisión	25°C		±4%HR	
	0-50°C			±5%HR
Intercambiabilidad	Totalmente intercambiables			
Rango de medición	0°C	30% HR		90%HR
	25°C	20% HR		90%HR
	50°C	20% HR		80%HR
Tiempo de respuesta (segundos)	1/e(63%)25°C, 1m/s Aire	6 S	10 S	15 S
Histéresis			±1%HR	
A largo plazo Estabilidad	Típico		±1%RH/año	
Temperatura				
Resolución		1°C	1°C	1°C
		8 bits	8 bits	8 bits
Repetibilidad			±1°C	
Precisión		±1°C		±2°C
Medición Gama		0°C		50°C
Tiempo de respuesta (Segundos)	1/e(63%)	6 S		30 S

Tabla 1: Características Sensores DHT11.
Fuente: DHT11 Humidity & Temperature Sensor.

Electricas

VDD=5V, T= 25°C (salvo indicación contraria)

	Condiciones	Mínimo	Típico	Máximo
Fuente de alimentación	DC	3V	5V	5.5V
Actual Suministro	Medición	0,5 mA		2,5 mA
	Media	0,2 mA		1 mA
	En espera	100uA		150uA
Muestreo periodo	Segundo	1		

Tabla 2: Características Eléctricas sensor DHT11.
Fuente: DHT11 Humidity & Temperature Sensor.

Anexo 2: Código Arduino para Lectura del Sensor DHT11

```
#include <DHT.h>  // Incluir la librería DHT
// Definir el pin donde está conectado el DHT11 y el tipo de sensor
#define DHTPIN 2     // Pin digital (ej. D2)
#define DHTTYPE DHT11 // Tipo DHT11
// Inicializar el sensor DHT
DHT dht(DHTPIN, DHTTYPE);

void setup() {
  Serial.begin(9600); // Iniciar comunicación serial
  dht.begin();        // Iniciar el sensor
  Serial.println("Lectura del sensor DHT11");
}
void loop() {
  delay(2000); // Esperar 2 segundos entre lecturas (el DHT11 es lento)

  // Leer humedad
  float humedad = dht.readHumidity();
  // Leer temperatura en Celsius
  float temperatura = dht.readTemperature();

  // Verificar si la lectura falló
  if (isnan(humedad) || isnan(temperatura)) {
    Serial.println("Error al leer el sensor DHT11");
    return;
  }
  // Mostrar datos en el monitor serial
  Serial.print("Humedad: ");
  Serial.print(humedad);
  Serial.print("%\t");
  Serial.print("Temperatura: ");
  Serial.print(temperatura);
  Serial.println("°C");
}
```

Anexo 3: Características técnicas Sensor de humedad tierra con LM393

Sonda HL-69

Parametro	Valor
Tipo de sensor	Resistivo
Rango de Resistencia	~1 kΩ (húmedo) a ~50 kΩ (seco) (varía según el suelo)
Voltaje de operación	3.3V - 5V DC
Corriente de trabajo	20mA
Material de electrodos	Acero inoxidable
Longitud	6 cm
Conexión	cable de 2 hilos

Tabla 3: Especificaciones técnicas sensor de humedad tierra.
Fuente: Elaboración propia.

Módulo LM393

Parametro	Valor
Chip principal	LM393
Voltaje de operación	3.3V a 5V
Consumo de corriente	0.8mA(max)
Rango de voltaje de entrada (IN+/IN-)	0V a 36V (limitado a 5V para trabajo en TTL)
Tiempo de respuesta	1.3 **µs**
Precision de comparación	±5 mV
Corriente de salida	20mA

Tabla 4: Especificaciones técnicas Módulo LM393..
Fuente: Elaboración propia.

Anexo 4: Código Arduino para Lectura del Sensor de suelo HL-69

```
const int sensorPin = A0;  // Pin analógico para el sensor HL-69
int humedadSuelo = 0;         // Variable para almacenar el valor de humedad

void setup() {
  Serial.begin(9600);      // Iniciar comunicación serial
}

void loop() {
   humedadSuelo = analogRead(sensorPin);   // Leer el valor analógico (0-1023)

   // Mostrar el valor crudo y porcentaje (ajusta los valores según calibración)
  Serial.print("Valor crudo: ");
  Serial.print(humedadSuelo);
  Serial.print(" | Humedad del suelo: ");
  Serial.print(convertirAPorcentaje(humedadSuelo));
  Serial.println("%");

  delay(2000);  // Esperar 2 segundos entre lecturas
}

// Función para convertir el valor analógico a porcentaje (ajusta los límites)
int convertirAPorcentaje(int valor) {
  // Valores de calibración (ajústalos según pruebas):
  int sueloSeco = 1023;    // Valor en aire seco
  int sueloMojado = 300;   // Valor en agua (puede variar)

  // Mapear el valor inversamente (mayor valor = menos humedad)
  int porcentaje = map(valor, sueloSeco, sueloMojado, 0, 100);

  // Limitar el rango entre 0% y 100%
  porcentaje = constrain(porcentaje, 0, 100);
  return porcentaje;
}
```

Anexo 5: Características técnicas Mosfet IRF520

Parametro	**Especificación**
MOSFET Principal	IRF520N
Voltaje de Operación	3.3V - 5V (aunque requiere 4V para activar completamente)
Voltaje de carga	24V
Voltaje Máximo (Vds)	100V
Corriente Máxima (Id)	9.7A (con disipador) 2A (sin disipador)
Resistencia ON	0.27Ω
Potencia Máxima	60W (con disipador)
Temperatura de trabajo	-55ºC a 175ºC

Tabla 5: Especificaciones técnicas Módulo Mosfet IRF520.
Fuente: Elaboración propia.

Anexo 6: Código Arduino para Activación Mosfet IRF520

```
const int botonPin = 2;       // Pin del botón conectado a GND (sin
resistencia externa)
const int mosfetPin = 3;    // Pin del MOSFET (Gate)
bool estadoCarga = false;   // Estado de la carga (ON/OFF)

void setup() {
   pinMode(botonPin, INPUT_PULLUP);   // Activar resistencia interna
PULL-UP
  pinMode(mosfetPin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
   int estadoBoton = digitalRead(botonPin);   // Leer el botón (LOW
cuando se presiona)

  if (estadoBoton == LOW) {                // Botón presionado (LOW por
PULL-UP)
    estadoCarga = !estadoCarga;            // Cambiar estado (toggle)
    digitalWrite(mosfetPin, estadoCarga);
         Serial.println(estadoCarga ? "MOSFET ACTIVADO" : "MOSFET
DESACTIVADO");
```

```
    delay(300);  // Anti-rebote
  }
}
```

Anexo 7: Características técnicas nRF24L01+

Operating conditions	Minimum	Maximum	Units
Supply voltages			
VDD	-0.3	3.6	V
VSS		0	V
Input voltage			
V_I	-0.3	5.25	V
Output voltage			
V_O	VSS to VDD	VSS to VDD	
Total Power Dissipation			
P_D (T_A=85°C)		60	mW
Temperatures			
Operating Temperature	-40	+85	°C
Storage Temperature	-40	+125	°C

Tabla 6: Características técnicas nRF24L01+..
Fuente: nRF24L01+ Product Specification v1.0

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
	Idle modes					
I_{VDD_PD}	Supply current in power down			900		nA
I_{VDD_ST1}	Supply current in standby-I mode	a		26		µA
I_{VDD_ST2}	Supply current in standby-II mode			320		µA
I_{VDD_SU}	Average current during 1.5ms crystal oscillator startup			400		µA
	Transmit					
I_{VDD_TX0}	Supply current @ 0dBm output power	b		11.3		mA
I_{VDD_TX6}	Supply current @ -6dBm output power	b		9.0		mA
I_{VDD_TX12}	Supply current @ -12dBm output power	b		7.5		mA
I_{VDD_TX18}	Supply current @ -18dBm output power	b		7.0		mA
I_{VDD_AVG}	Average Supply current @ -6dBm output power, ShockBurst™	c		0.12		mA
I_{VDD_TXS}	Average current during TX settling	d		8.0		mA
	Receive					
I_{VDD_2M}	Supply current 2Mbps			13.5		mA
I_{VDD_1M}	Supply current 1Mbps			13.1		mA
I_{VDD_250}	Supply current 250kbps			12.6		mA
I_{VDD_RXS}	Average current during RX settling	e		8.9		mA

Tabla 7: Especificaciones Eléctricas nRF24L01+.
Fuente: nRF24L01+ Product Specification v1.0

Parámetro	Especificación
Protocolo	Propietario (compatible con IEEE 802.15.4)
Rango de Frecuencia	2.400 - 2.525 GHz (126 canales de 1 MHz)
Potencia de Transmisión	-18 dBm a 0 dBm (ajustable en 4 niveles)
Velocidades	250 kbps, 1 Mbps, 2 Mbps
Sensibilidad	-82 dBm (a 2 Mbps), -94 dBm (a 250 kbps)
Direcciones	hasta 6 pipes de recepción
Retransmisiones	Hasta 15 intentos automáticos (con timeout configurable)
Tiempo de Cambio TX/RX	130 µs
Interfaz	SPI
Alcance	100m (max)

Tabla 8: Valores Técnicos del módulo nRF24L01+.
Fuente: Elaboración propia.

Anexo 8: Código Arduino para la transmisión de datos del nRF24L01

Recepción

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

RF24 radio(9, 8); // CE, CSN

// Direcciones de comunicación (deben coincidir en TX y RX)
const byte address[6] = "00001";

void setup() {
  Serial.begin(9600);
  radio.begin();
  radio.openWritingPipe(address); // Dirección para enviar datos
```

```
    radio.openReadingPipe(1,  address);  //  Dirección  para  recibir
confirmación
  radio.setPALevel(RF24_PA_MIN); // Ajusta la potencia (MIN, LOW, HIGH,
MAX)
  radio.stopListening(); // Modo transmisor
}

void loop() {
  const char text[] = "Hola"; // Mensaje a enviar
  radio.write(&text, sizeof(text)); // Enviar mensaje
  Serial.print("Enviado: ");
  Serial.println(text);

  // Esperar confirmación
  radio.startListening();
  unsigned long startTime = millis();
  bool timeout = false;
  char ack[10] = "";

  while (!radio.available() && !timeout) {
    if (millis() - startTime > 1000) { // Timeout de 1 segundo
      timeout = true;
      Serial.println("Esperando confirmación...");
    }
  }

  if (!timeout) {
    radio.read(&ack, sizeof(ack)); // Leer confirmación
    Serial.print("Confirmación: ");
    Serial.println(ack);
  }

  radio.stopListening();
  delay(2000); // Esperar 2 segundos entre envíos
}
```

Transmisión

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>

RF24 radio(9, 8); // CE, CSN
```

```
// Dirección (debe coincidir con el TX)
const byte address[6] = "00001";

void setup() {
  Serial.begin(9600);
  radio.begin();
  radio.openReadingPipe(0, address); // Dirección para recibir datos
  radio.openWritingPipe(address); // Dirección para enviar confirmación
  radio.setPALevel(RF24_PA_MIN); // Misma potencia que el TX
  radio.startListening(); // Modo receptor
}

void loop() {
  if (radio.available()) {
    char text[10] = "";
    radio.read(&text, sizeof(text)); // Leer mensaje
    Serial.print("Recibido: ");
    Serial.println(text);

    // Enviar confirmación
    const char ack[] = "Recibido";
    radio.stopListening();
    radio.write(&ack, sizeof(ack));
    radio.startListening();
    Serial.println("Confirmación enviada");
  }
}
```

Anexo 9: Características técnicas LCD16x2 + Modulo I2C

Parametro	**Especificación**
Tipo de display	LCD alfanumérico de matriz de puntos
Tampó de caracteres	5×8 píxeles (estándar), 5×10 píxeles (especiales)
Tecnología	STN (Super Twisted Nematic) con polarizador.
Voltaje de operación	4.5V - 5.5V
Corriente de operación	1.0 - 2.0 mA

Corriente retroiluminación	20 - 60 mA
Consumo en standby	< 0.1 mA
Luminancia	70 - 120 cd/m²
Ángulo de visión	±45° horizontal, ±30° vertical
Temperatura operación	0ºC a 50ºC
Temperatura almacenamiento	-20ºC a 75ºC
Humedad relativa	20% a 80% (sin condensación)

Tabla 9: Características técnicas Pantalla LCD.
Fuente: Elaboración propia.

Anexo 10: Código Arduino para el funcionamiento pantalla LCD con módulo I2C:

```
#include <Wire.h>           // Librería para comunicación I2C
#include <LiquidCrystal_I2C.h> // Librería para LCD con I2C

// Inicializa la pantalla LCD (0x27 es la dirección común, puede
variar)
LiquidCrystal_I2C lcd(0x27, 16, 2); // (Dirección I2C, columnas, filas)

void setup() {
  lcd.init();           // Inicializa el LCD
  lcd.backlight();      // Enciende la retroiluminación

  // Mensaje inicial
  lcd.setCursor(0, 0); // (Columna, Fila)
  lcd.print("Hola Mundo!");

  lcd.setCursor(0, 1);
  lcd.print("LCD con I2C");
}

void loop() {

  //enseña el tiempo desde el encendido
  lcd.setCursor(0, 1);
  lcd.print("T: ");
  lcd.print(millis() / 1000);
  lcd.print(" seg");
```

```
  delay(1000);
}
```

Anexo 11: Características técnicas Bomba de agua sumergible 5V

Parametro	**Especificación**
Voltaje de operación	2.5V a 6V
Corriente de Operación	130 a 220mA
Flujo	Flujo : 80 ~ 120 L/H
Elevación máxima de columna de agua	40 centímetros
Material	Plástico
Diámetro Externo del Tubo de Salida	7.5 mm
Diámetro Interno del Tubo de Salida	5 mm
Largo de cable	20 cm
Tamaño	43mm (largo) x 23mm (diámetro)
Temperatura del agua	-20 a 50 °C

Tabla 10: Características técnica Bomba de agua sumergible.
Fuente: Elaboración propia.

Anexo 12: Código de Arduino Uno

```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <DHT.h>

#define CE_PIN   9
#define CSN_PIN 10
#define sensor1 A0
#define sensor2 A1
#define Mosfet_PIN 2
#define Led_Pin 4
#define DHTPIN 3
```

```
#define DHTTYPE DHT11


const byte slaveAddress[5] = {'R','x','A','A','A'};
const byte thisMasterAddress[5] = {'T','x','A','A','A'};

RF24 radio(CE_PIN, CSN_PIN);
DHT dht(DHTPIN, DHTTYPE);

struct SensorData {
  int humedadSuelo1;
  int humedadSuelo2;
  float temperatura;
  float humedadAire;
};

char dataReceived[32] = {0};
int umbralRecibido = 0;
bool nuevoUmbral = false;

unsigned long currentMillis;
unsigned long prevSensorRead = 0;
const unsigned long sensorInterval = 2000;
unsigned long prevTxMillis = 0;
const unsigned long txInterval = 2000;
unsigned long MosfetStartTime = 0;
const unsigned long MosfetDuracion = 10000;
bool MosfetActiva = false;

// Variables renombradas consistentemente
int lastHumedadSuelo1 = 0;
int lastHumedadSuelo2 = 0;
float lastTemperatura = 0;
float lastHumedadAire = 0;

void setup() {
    Serial.begin(115200);
    while (!Serial);
    Serial.println("Iniciando Maestro con DHT11");

    dht.begin();

    if (!radio.begin()) {
```

```
        Serial.println("Error: Módulo RF24 no detectado!");
        while (1);
    }

    radio.setChannel(108);
    radio.setDataRate(RF24_250KBPS);
    radio.setRetries(5, 15);
    radio.setPALevel(RF24_PA_LOW);
    radio.openWritingPipe(slaveAddress);
    radio.openReadingPipe(1, thisMasterAddress);
    radio.startListening();

    pinMode(Mosfet_PIN, OUTPUT);
    digitalWrite(Mosfet_PIN, LOW);
    pinMode(Led_Pin, OUTPUT);
    digitalWrite(Led_Pin, LOW);
    leerSensores();
}

void loop() {
    currentMillis = millis();

    if (currentMillis - prevSensorRead >= sensorInterval) {
        prevSensorRead = currentMillis;
        leerSensores();
    }

    if (currentMillis - prevTxMillis >= txInterval) {
        prevTxMillis = currentMillis;
        enviarDatosSensores();
    }

    recibirDatosControl();
    controlBomba();
}

void leerSensores() {
    // Humedad del suelo
    int sumaH1 = 0, sumaH2 = 0;
    const int muestras = 5;

    for(int i = 0; i < muestras; i++) {
        sumaH1 += analogRead(sensor1);
```

```
        sumaH2 += analogRead(sensor2);
        delay(50);
    }

    lastHumedadSuelo1 = map(sumaH1/muestras, 0, 1023, 100, 0);
    lastHumedadSuelo2 = map(sumaH2/muestras, 0, 1023, 100, 0);

    // Lectura DHT11 con verificación de errores CORREGIDA
    float h = dht.readHumidity();
    float t = dht.readTemperature();

    if (!isnan(h)) { lastHumedadAire = h; }
    if (!isnan(t)) { lastTemperatura = t; }

    Serial.print("Lecturas - S1:");
    Serial.print(lastHumedadSuelo1);
    Serial.print("%, S2:");
    Serial.print(lastHumedadSuelo2);
    Serial.print("%, Aire:");
    Serial.print(lastHumedadAire);
    Serial.print("%, Temp:");
    Serial.print(lastTemperatura);
    Serial.println("°C");
}

void enviarDatosSensores() {
    radio.stopListening();
    delay(10);

    SensorData datos;
    datos.humedadSuelo1 = lastHumedadSuelo1;
    datos.humedadSuelo2 = lastHumedadSuelo2;
    datos.temperatura = lastTemperatura;
    datos.humedadAire = lastHumedadAire;

    bool resultado = radio.write(&datos, sizeof(datos));
    Serial.print("Enviando - S1:");
    Serial.print(datos.humedadSuelo1);
    Serial.print("%, S2:");
    Serial.print(datos.humedadSuelo2);
    Serial.print("%, Aire:");
    Serial.print(datos.humedadAire);
    Serial.print("%, Temp:");
```

```
    Serial.print(datos.temperatura);
    Serial.println(resultado ? "°C -> OK" : "°C -> Fallo");

    radio.startListening();
}

void recibirDatosControl() {
    if (radio.available()) {
        memset(dataReceived, 0, sizeof(dataReceived));
        radio.read(&dataReceived, sizeof(dataReceived));

        if (strncmp(dataReceived, "u:", 2) == 0) {
            umbralRecibido = atoi(dataReceived + 2);
            nuevoUmbral = true;

            Serial.print("Umbral recibido: ");
            Serial.print(umbralRecibido);
            Serial.println("%");
        }
    }
}

void controlBomba() {
    if (nuevoUmbral) {
        nuevoUmbral = false;

                bool debeActivar = (umbralRecibido > lastHumedadSuelo1)
|| (umbralRecibido > lastHumedadSuelo2);

        if (debeActivar && !MosfetActiva) {
            digitalWrite(Mosfet_PIN, HIGH);
            digitalWrite(Led_Pin, HIGH);
            MosfetStartTime = currentMillis;
            MosfetActiva = true;
            Serial.println("Bomba ACTIVADA por 10 segundos");
        }
    }

         if  (MosfetActiva  &&  (currentMillis  -  MosfetStartTime  >=
MosfetDuracion)) {
        digitalWrite(Mosfet_PIN, LOW);
        digitalWrite(Led_Pin, LOW);
        MosfetActiva = false;
```

```
        Serial.println("Bomba DESACTIVADA (tiempo completado)");
    }
}
```

Anexo 13: Código del ESP32

```
#include <SPI.h>
#include <RF24.h>
#include <nRF24L01.h>
#include <WiFi.h>
#include <WebServer.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>


#define CE_PIN 22
#define CSN_PIN 21
#define thresholdPin 36    // Potenciómetro
#define TOUCH_PIN_1 14     // Touch Pin para 30%
#define TOUCH_PIN_2 27    // Touch Pin para 70%
#define TOUCH_PIN_MAN 32   // Touch Pin para modo manual
#define touchNav_Pin 33   //Touch Pin Navegar por LCD
#define LCD_SDA 16
#define LCD_SCL 17

// Configuración WiFi
const char* ssid = "McMarciano";
const char* password = "McMa1503";

// Configuración radio
const byte thisSlaveAddress[5] = {'R','x','A','A','A'};
const byte masterAddress[5] = {'T','x','A','A','A'};

LiquidCrystal_I2C lcd(0x27,16,2);
RF24 radio(CE_PIN, CSN_PIN);
WebServer server(80);

// Variables globales
int potValue = 0;
```

```
int umbral = 0;
int  touchThreshold  =  35;    //  Sensibilidad  touch  (ajustar  según
necesidad)
bool manualMode = true;   // Inicia en modo manual
char dataToSend[32] = "ACK-Eslavo";
unsigned long lastTouchTime = 0;
const long touchDebounce = 500;  // Tiempo anti-rebote en ms
// Variables para navegación LCD
int lcdScreen = 0;
bool buttonReleased = true;
//Almacenar datos enviados del arduino, mayor optimizacion

// Estructura para recibir datos (DEBE COINCIDIR con el Arduino)
struct SensorData
  {
    int16_t humedadSuelo1;
    int16_t humedadSuelo2;
    float temperatura;
    float humedadAire;
  };
SensorData newData;



// Funciones
void handleRoot();
void handleSetMode();
void handleGetData();
void enviarUmbral();
void getData();
void botonNavegador();
void initWiFi();
void checkTouchPins();
void ActualizarUmbral();
void ActualizacionLCD();
String getWebPage();

void setup()
 {
    Serial.begin(115200);

    // Configurar sensibilidad de los touch pins
    touchAttachInterrupt(TOUCH_PIN_1, [](){}, touchThreshold);
```

```
    touchAttachInterrupt(TOUCH_PIN_2, [](){}, touchThreshold);
    touchAttachInterrupt(TOUCH_PIN_MAN, [](){}, touchThreshold);
    touchAttachInterrupt(touchNav_Pin, [](){}, touchThreshold);
    // Inicializar LCD
    Wire.begin(LCD_SDA, LCD_SCL); //Pines personalizados
    lcd.init();
    lcd.backlight();
    lcd.clear();
    lcd.print("Iniciando...");
    // Inicializar radio
    if (!radio.begin())
      {
        Serial.println("Error: Módulo RF24 no detectado!");
        while (1);
      }

    // Configurar radio
    radio.setChannel(108);
    radio.setDataRate(RF24_250KBPS);
    radio.setRetries(5, 15);
    radio.setPALevel(RF24_PA_LOW);
    radio.openReadingPipe(1, thisSlaveAddress);
    radio.openWritingPipe(masterAddress);
    radio.startListening();

    // Inicialización y Configuración rutas del servidor web
    initWiFi();
    server.on("/", handleRoot);
    server.on("/setMode", handleSetMode);
    server.on("/getData", handleGetData);
    server.begin();
    Serial.println("Sistema iniciado");
    Serial.print("Entrar en la Web con IP");
    Serial.println(WiFi.localIP());
}

void loop()
{
    server.handleClient();
    checkTouchPins();
    ActualizarUmbral();
    enviarUmbral();
    getData();
```

```
    botonNavegador();
    ActualizacionLCD();
    delay(10);
 }

void initWiFi()
  {
    Serial.println();
    Serial.print("Conectando a ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);

    unsigned long startAttemptTime = millis();
     while (WiFi.status() != WL_CONNECTED && millis() - startAttemptTime
< 10000)
    {
     delay(500);
     Serial.print(".");
    }

    if (WiFi.status() == WL_CONNECTED)
    {
      Serial.println("\nWiFi conectado");
      Serial.print("IP: ");
      Serial.println(WiFi.localIP());
    } else {
     Serial.println("\nError WiFi");
    }
  }

void checkTouchPins() {
    if (millis() - lastTouchTime < touchDebounce) return;
    if (touchRead(TOUCH_PIN_1) < touchThreshold) {
        manualMode = false;
        umbral = 30;
        lastTouchTime = millis();
        lcd.setCursor(0,0);
        lcd.print("Suelo poco humedo");
    }
    else if (touchRead(TOUCH_PIN_2) < touchThreshold) {
        manualMode = false;
        umbral = 70;
        lastTouchTime = millis();
```

```
        lcd.setCursor(0,0);
        lcd.print("Suelo casi humedo");
    }
    else if (touchRead(TOUCH_PIN_MAN) < touchThreshold) {
        manualMode = true;
        lastTouchTime = millis();
        lcd.setCursor(0,0);
        lcd.print("Modo manual activado");
    }
}

void ActualizarUmbral() {
    if (manualMode) {
        int potValue = analogRead(thresholdPin);
        umbral = map(potValue, 0, 4095, 0, 100);
    }
}

void ActualizacionLCD() {
    static unsigned long lastUpdate = 0;
    if(millis() - lastUpdate < 500) return;
    lcd.setCursor(0, 0);

    switch(lcdScreen) {
        case 0: // Pantalla principal
        {
            lcd.clear();
            lcd.print("Humedad 1: ");
            lcd.print(newData.humedadSuelo1);
            lcd.print("%");
            lcd.setCursor(0,1);
            lcd.print("Humedad 2: ");
            lcd.print(newData.humedadSuelo2);
            lcd.print("%");

            break;
        }
        case 1:
        {
            lcd.clear();
            lcd.print("Humedad Aire:");
            lcd.print(newData.humedadAire, 0);
```

```
            lcd.print("%");
            lcd.setCursor(0, 1);
            lcd.print("Temperatura: ");
            lcd.print(newData.temperatura, 1);
            lcd.print((char)223);
            lcd.print("°C");
            break;
        }
        case 2: // Pantalla de configuración
         {
            lcd.clear();
            lcd.print("Umbral en:");
             lcd.print(umbral);
             lcd.print("%");
            lcd.setCursor(0, 1);
            lcd.print("Programa: ");
                 lcd.print(umbral == 30 ? "1" : umbral == 70 ? "2" :
"Manual");
            lcd.print("  ");
            break;
         }
        case 3: // Pantalla de conexión
         {
            lcd.clear();
            lcd.print("La Web con IP");
            lcd.setCursor(0, 1);
            lcd.print(WiFi.localIP());
         }
    }
      lastUpdate = millis();
}
void botonNavegador()
 {
    const int touchNav = touchRead(touchNav_Pin);

    if(touchNav < touchThreshold) { // Botón presionado
        static unsigned long lastTouchTime = 0;
        if (millis() - lastTouchTime > 900) {
        lcdScreen = (lcdScreen + 1) % 4; // Rotar entre 0, 1, 2
        lcd.clear();
        lastTouchTime = millis();
    }
    }
```

```
}

void enviarUmbral() {
    static unsigned long lastSend = 0;
    static int lastUmbral = -1;

    if (umbral != lastUmbral || millis() - lastSend > 500) {
        radio.stopListening();
        delay(10);
        snprintf(dataToSend, sizeof(dataToSend), "u:%d", umbral);
        bool success = radio.write(&dataToSend, sizeof(dataToSend));
        if (success) {
            Serial.print("Umbral enviado: ");
            Serial.print(umbral);
            Serial.println(manualMode ? "% (Manual)" : "% (Preset)");
            lastUmbral = umbral;
            lastSend = millis();
        } else {
            Serial.println("Error al enviar");
        }
        radio.startListening();
    }
}

void getData() {
    if (radio.available()) {
        radio.read(&newData, sizeof(newData));
        Serial.print("Datos recibidos - S1:");
        Serial.print(newData.humedadSuelo1);
        Serial.print("%, S2:");
        Serial.print(newData.humedadSuelo2);
        Serial.print("%, Aire:");
        Serial.print(newData.humedadAire);
        Serial.print("%, Temp:");
        Serial.print(newData.temperatura);
        Serial.println("°C");
    }
}

String getWebPage() {
         String  page  =  "<!DOCTYPE  html><html><head><title>McGarden
Labs</title>";
```

```
        page += "<meta name='viewport' content='width=device-width,
initial-scale=1'>";
    page += "<meta http-equiv='refresh' content='2'>"; // Actualización
cada 2 segundos
                        page      +=      "<link      rel='stylesheet'
href='https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all
.min.css'>";
    page += "<style>";
    page += ":root {";
    page += "  --primary: #4CAF50;";
    page += "  --secondary: #2196F3;";
    page += "  --warning: #ff9800;";
    page += "  --card-bg: #ffffff;";
    page += "  --text-color: #333;";
    page += "}";
    page += "body {";
      page += "   font-family: 'Segoe UI', Tahoma, Geneva, Verdana,
sans-serif;";
    page += "  margin: 0; padding: 20px;";
    page += "  background-color: #f5f5f5;";
    page += "  color: var(--text-color);";
    page += "}";
    page += ".container {";
    page += "  max-width: 1000px; margin: 0 auto;";
    page += "  background: white; padding: 20px;";
        page += "    border-radius: 10px; box-shadow: 0 2px 10px
rgba(0,0,0,0.1);";
    page += "}";
    page += "h1 { color: var(--primary); text-align: center; }";
    page += ".sensor-grid {";
    page += "  display: grid;";
     page += "   grid-template-columns: repeat(auto-fit, minmax(240px,
1fr));";
    page += "  gap: 15px; margin-bottom: 25px;";
    page += "}";
    page += ".sensor-card {";
    page += "  background: var(--card-bg); border-radius: 8px;";
    page += "  padding: 15px; box-shadow: 0 2px 5px rgba(0,0,0,0.1);";
    page += "  border-left: 4px solid var(--primary);";
    page += "}";
    page += ".sensor-value {";
    page += "  font-size: 28px; font-weight: bold;";
    page += "  text-align: center; margin: 10px 0;";
```

```
    page += "  color: var(--primary);";
    page += "}";
    page += ".progress-container { margin: 25px 0; }";
    page += ".progress {";
    page += "  width: 100%; height: 20px;";
    page += "  background-color: #e0e0e0; border-radius: 10px;";
    page += "  overflow: hidden;";
    page += "}";
    page += ".progress-bar {";
    page += "  height: 100%;";
      page += "  background: linear-gradient(90deg, var(--primary), #8BC34A);";
    page += "  border-radius: 10px;";
     page += "  display: flex; align-items: center; justify-content: center;";
    page += "  color: white; font-size: 12px; font-weight: bold;";
    page += "}";
    page += ".btn-group {";
    page += "  display: flex; flex-wrap: wrap; gap: 10px;";
    page += "  margin: 20px 0;";
    page += "}";
    page += ".btn {";
    page += "  padding: 10px 15px; border: none; border-radius: 5px;";
    page += "  cursor: pointer; font-weight: bold;";
     page += "  display: flex; align-items: center; justify-content: center;";
    page += "}";
    page += ".btn i { margin-right: 8px; }";
    page += ".btn-preset {";
     page += "  background: linear-gradient(135deg, var(--secondary), #03A9F4);";
    page += "  color: white;";
    page += "}";
    page += ".btn-manual {";
      page += "  background: linear-gradient(135deg, var(--warning), #FFC107);";
    page += "  color: white;";
    page += "}";
    page += ".active {";
    page += "  box-shadow: 0 0 0 3px rgba(0,0,0,0.2);";
    page += "}";
    page += "@media (max-width: 600px) {";
    page += "  .btn-group { flex-direction: column; }";
```

```
    page += "  .btn { width: 100%; }";
    page += "}";
    page += "</style>";
    page += "</head><body>";
    page += "<div class='container'>";
    page += "<h1><i class='fas fa-seedling'></i> McGarden Labs</h1>";

    // Sensores
    page += "<div class='sensor-grid'>";
    page += "<div class='sensor-card'>";
    page += "<h2><i class='fas fa-tint'></i> Humedad Suelo 1</h2>";
          page  +=  "<div  id='soil1-value'  class='sensor-value'>"  +
String(newData.humedadSuelo1) + "%</div>";
    page += "</div>";

    page += "<div class='sensor-card'>";
    page += "<h2><i class='fas fa-tint'></i> Humedad Suelo 2</h2>";
          page  +=  "<div  id='soil2-value'  class='sensor-value'>"  +
String(newData.humedadSuelo2) + "%</div>";
    page += "</div>";

    page += "<div class='sensor-card'>";
    page += "<h2><i class='fas fa-cloud'></i> Humedad Aire</h2>";
           page  +=  "<div  id='air-value'  class='sensor-value'>"  +
String(newData.humedadAire, 0) + "%</div>";
    page += "</div>";

    page += "<div class='sensor-card'>";
            page  +=  "<h2><i  class='fas  fa-thermometer-half'></i>
Temperatura</h2>";
          page  +=  "<div  id='temp-value'  class='sensor-value'>"  +
String(newData.temperatura, 1) + "C</div>";
    page += "</div>";
    page += "</div>";

    // Progreso
    page += "<div class='progress-container'>";
            page  +=  "<div  class='progress-label'><span>Umbral  de
humedad:</span>  <span  id='umbral-value'>"  +  String(umbral)  +
"%</span></div>";
           page  +=  "<div  class='progress'><div  id='progress-bar'
class='progress-bar'  style='width:"  +  String(umbral)  +  "%'>"  +
String(umbral) + "%</div></div>";
```

```
    page += "</div>";

    // Botones
    page += "<h2><i class='fas fa-cogs'></i> Modo de Operacion</h2>";
    page += "<div class='btn-group'>";
    page += "<button id='preset30' class='btn btn-preset";
    if (umbral == 30 && !manualMode) page += " active";
          page  +=  "'  onclick='setMode(\"preset30\")'><i  class='fas
fa-leaf'></i> Suelo poco humedo</button>";

    page += "<button id='preset70' class='btn btn-preset";
    if (umbral == 70 && !manualMode) page += " active";
          page  +=  "'  onclick='setMode(\"preset70\")'><i  class='fas
fa-tree'></i> suelo casi humedo</button>";

    page += "<button id='manual' class='btn btn-manual";
    if (manualMode) page += " active";
          page  +=  "'  onclick='setMode(\"manual\")'><i  class='fas
fa-sliders-h'></i> Manual</button>";
    page += "</div>";

    // JavaScript
    page += "<script>";
    page += "function setMode(mode) {";
       page  +=  "    fetch('/setMode?mode='  +  mode).then(response  =>
response.text()).then(data => {";
    page += "    console.log('Mode set:', data);";
    page += "    updateStatus();";
    page += "  });";
    page += "}";

    page += "function updateStatus() {";
             page   +=   "       fetch('/getData').then(response   =>
response.text()).then(data => {";
    page += "    const parts = data.split(',');";
      page  +=  "     document.getElementById('soil1-value').innerText  =
parts[0] + '%';";
      page  +=  "     document.getElementById('soil2-value').innerText  =
parts[1] + '%';";
      page  +=  "      document.getElementById('air-value').innerText  =
parts[2] + '%';";
      page  +=  "     document.getElementById('temp-value').innerText  =
parts[3] + '°C';";
```

```
    page += "  });";
    page += "}";

      page += "// Actualizar cada 2 segundos (complementario al meta
refresh)";
    page += "setInterval(updateStatus, 2000);";
    page += "</script>";

    page += "</div></body></html>";

    return page;
}

void handleRoot() {
    server.send(200, "text/html", getWebPage());
}

void handleSetMode() {
    if (server.hasArg("mode")) {
        String mode = server.arg("mode");

        if (mode == "preset30") {
            manualMode = false;
            umbral = 30;
            server.send(200, "text/plain", "Preset 30%");
        } else if (mode == "preset70") {
            manualMode = false;
            umbral = 70;
            server.send(200, "text/plain", "Preset 70%");
        } else if (mode == "manual") {
            manualMode = true;
            server.send(200, "text/plain", "Manual");
        } else {
            server.send(400, "text/plain", "Modo inválido");
        }
    } else {
        server.send(400, "text/plain", "Falta parámetro mode");
    }
}

void handleGetData() {
    String data = String(newData.humedadSuelo1) + "," +
                  String(newData.humedadSuelo2) + "," +
```

```
                String(newData.humedadAire, 0) + "," +
                String(newData.temperatura, 1);
    server.send(200, "text/plain", data);
}
```