UNIVERSIDAD PRIVADA DE TACNA
FACULTAD DE INGENIERÍA
ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

TESIS

"DISEÑO DE UN PROTOTIPO DE RIEGO AUTOMATIZADO CON MONITOREO VÍA IOT PARA MITIGAR LOS EFECTOS DE LAS HELADAS EN CULTIVOS DE ZONAS ALTOANDINAS, 2025"

PARA OPTAR:

TÍTULO PROFESIONAL DE INGENIERO ELECTRÓNICO

PRESENTADO POR:

Bach. ANGEL ROMARIO CCALLOMAMANI CCALLOMAMANI

Bach. GERSON JEANPAUL VILLA FLORES

TACNA – PERÚ

2025

UNIVERSIDAD PRIVADA DE TACNA
FACULTAD DE INGENIERÍA
ESCUELA PROFESIONAL DE INGENIERÍA ELECTRÓNICA

TESIS

"DISEÑO DE UN PROTOTIPO DE RIEGO AUTOMATIZADO CON MONITOREO VÍA IOT PARA MITIGAR LOS EFECTOS DE LAS HELADAS EN CULTIVOS DE ZONAS ALTOANDINAS, 2025"

Tesis sustentada y aprobada el 11 de diciembre de 2025; estando el jurado calificador integrado por:

PRESIDENTE	**:**	**Mtro. MARIA ELENA VILDOZO ZAMBRANO**
SECRETARIO	**:**	**Mtro. CARLOS ARMANDO RODRIGUEZ SILVA**
VOCAL	**:**	**Msc. MILAGROS GLENY COHAILA GONZALES**
ASESOR	**:**	**Mag. TITO LEONCIO CÓRDOVA MIRANDA**

DECLARACIÓN JURADA DE ORIGINALIDAD

Nosotros, Angel Romario Ccallomamani Ccallomamani y Gerson Jeanpaul Villa Flores, egresados de la Escuela Profesional de Ingeniería Electrónica de la Facultad de Ingeniería de la Universidad Privada de Tacna, identificados con DNI 71654330 Y 71205414 respectivamente, así como Tito Leoncio Córdova Miranda con DNI 04643495; declaramos en calidad de autores y asesor que:

1. Somos los autores de la tesis titulada: *Diseño de un prototipo de riego automatizado con monitoreo vía IoT para mitigar los efectos de las heladas en cultivos de zonas altoandinas, 2025*, la cual presentamos para optar el Título Profesional de *Ingeniero Electrónico.*
2. La tesis es completamente original y no ha sido objeto de plagio, total ni parcialmente, habiéndose respetado rigurosamente las normas de citación y referencias para todas las fuentes consultadas.
3. Los datos presentados en los resultados son auténticos y no han sido objeto de manipulación, duplicación ni copia.

En virtud de lo expuesto, asumimos frente a *La Universidad* toda responsabilidad que pudiera derivarse de la autoría, originalidad y veracidad del contenido de la tesis, así como por los derechos asociados a la obra.

En consecuencia, nos comprometemos ante *La Universidad* y terceros a asumir cualquier perjuicio que pueda surgir como resultado del incumplimiento de lo aquí declarado, o que pudiera ser atribuido al contenido de la tesis, incluyendo cualquier obligación económica que debiera ser satisfecha a favor de terceros debido a acciones legales, reclamos o disputas resultantes del incumplimiento de esta declaración.

En caso de descubrirse fraude, piratería, plagio, falsificación o la existencia de una publicación previa de la obra, aceptamos todas las consecuencias y sanciones que puedan derivarse de nuestras acciones, acatando plenamente la normatividad vigente.

Tacna, 12 de agosto de 2025

Ángel Romario Ccallomamani Ccallomamani
DNI: 71654330

Gerson Jeanpaul Villa Flores
DNI: 71205414

Tito Leoncio Córdova Miranda
DNI: 04643495

DEDICATORIA

Este trabajo se lo dedico, con todo mi corazón, a mis padres Alfredo y Celina, por su amor, paciencia y confianza en mí; y a mis hermanos, por su apoyo incondicional y por recordarme que las metas son más valiosas cuando se comparten con quienes amas.

Angel Romario Ccallomamani Ccallomamani

Dedico esta tesis a mis padres, a mi hermano, a toda mi familia, que gracias a ellos es que puedo desarrollar mis metas y alcanzar mis objetivos.

Gerson Jeanpaul Villa Flores

AGRADECIMIENTO

A mi familia, por su apoyo constante y la motivación que me brindaron para continuar hasta alcanzar esta meta. A mis docentes, por su enseñanza, orientación y por compartir su experiencia con dedicación, lo que contribuyó de manera significativa a mi formación académica y personal.

Angel Romario Ccallomamani Ccallomamani

Agradecer a Dios por darme la vida y la capacidad para aprender todo lo que sé hasta ahora, también agradezco a mis padres por haberme enseñado desde pequeño que uno tiene que esforzarse para conseguir lo que quiere.

Gerson Jeanpaul Villa Flores

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE ANEXOS

RESUMEN

El objetivo de la presente tesis es diseñar un prototipo de riego automatizado con monitoreo vía IOT que ayude a mitigar los efectos de las heladas en las zonas altoandinas del Perú. Los parámetros principales usados la medición del fenómeno de la helada fueron: temperatura del ambiente, temperatura del suelo, humedad del suelo, velocidad de viento, nivel de radiación solar, adicional también se usó el nivel de agua para hacer el riego por aspersión. Teniendo estos parámetros en cuenta se adaptaron los respectivos sensores usando módulos de desarrollo ESP32 y un módulo Raspberry para que se almacenen los valores del historial de datos. Los datos obtenidos pueden ser observar localmente y remotamente mediante la interfaz Adafruit y puede ser visualizados desde cualquier dispositivo que tenga acceso e internet y también mediante notificaciones con la aplicación Telegram, en esta app los datos se envían cada vez que haya un cambio en los valores de los parámetros con sus respectivas alertas en caso de una próxima helada. Todo el desarrollo sirve para poder observar y tomar decisiones respecto a los cultivos en las zonas altoandinas y dio como resultado que las partes del prototipo resisten temperaturas bajo cero y el uso ayuda a reducir los efectos producidos por s heladas en los cultivos de zonas altoandinas.

Palabras Clave: Heladas; zonas altoandinas; internet de las cosas; riego automatizado; prototipo.

ABSTRACT

The objective of this thesis is to design an automated irrigation prototype with IoT monitoring that helps mitigate the effects of frost in the high Andean areas of Peru. The main parameters used to measure the frost phenomenon were: ambient temperature, soil temperature, soil moisture, wind speed, solar radiation level, and water level was also used for sprinkler irrigation. Taking these parameters into account, the respective sensors were adapted using ESP32 development modules and a Raspberry module to store historical data values. The data obtained can be observed locally and remotely through the Adafruit interface and can be viewed from any device with internet access. It can also be viewed through notifications with the Telegram application. In this app, data is sent every time there is a change in the parameter values, with their respective alerts in case of an upcoming frost. The entire development allows for observation and decision-making regarding crops in high Andean regions. The prototype's parts are resistant to sub-zero temperatures, and their use helps reduce the effects of frost on crops in high Andean regions.

Keywords: Frost; high Andean regions; Internet of Things; automated irrigation; prototype.

INTRODUCCIÓN

A lo largo de nuestra historia, el hombre ha luchado contra adversidades enormes tales como animales salvajes o incluso un lugar cómodo donde habitar, luego de solucionar estas adversidades se enfrenta a un reto más grande, encontrar su lugar entre él y la naturaleza. Así como la naturaleza da cosas también las quita, debido a esto el hombre ha tenido que adaptarse a los constantes cambios climáticos como el calor abrasador y el frío extremo. Debido a este concepto, tocaremos el tema de como el frío extremo afecta al hombre y a todo lo que lo rodea en su día a día.

Si nos enfocamos en el área de nuestro país Perú nos damos cuenta que los lugares donde el clima es más frío están ubicados en la “altura” o como es conocido: las zonas altoandinas. Existen varias comunidades en las zonas altoandinas donde el frío llega a superar los -10 ºC, incluso con ese nivel de frío la gente tiene plantas que cosechar para poder comer ellos mismos o darle el alimento a sus animales que los cobijan del frío.

El objetivo de esta tesis es poder acercarnos un poco más a esa realidad nuestra y buscar una solución frente a este problema de las heladas haciendo diseño de un prototipo de riego automatizado usando IOT para poder reducir al mínimo los efectos de las heladas en los mismos cultivos de las zonas altoandinas.

La presente tesis está estructurada de la siguiente manera:

Capítulo I: Problema de Investigación, donde el problema general es como diseñar un prototipo para prevenir el efecto de las heladas en zonas altoandinas usando IoT, también se dan a conocer los objetivos generales y objetivos específicos de la tesis.

Capítulo II: Marco Teórico, donde se desarrollan los antecedentes nacionales, internacionales y los fundamentos teóricos para el diseño del prototipo usando la tecnología IoT.

Capítulo III: Marco Metodológico, donde se da a conocer la metodología de la investigación y la propuesta de una mejor metodología para un diseño basado en tecnología IoT, también su implementación y sus respectivas pruebas.

Capítulo IV: Resultados, donde se da a conocer los resultados del diseño físico y diseño lógico, también fotografías del prototipo en funcionamiento.

Capítulo V: Discusión, donde se da a conocer la información para el diseño del prototipo IoT, su correcto armado y su funcionamiento.

Y, por último, se da a conocer la bibliografía y webgrafía utilizado en el desarrollo de la presente tesis, recomendaciones del prototipo, y sus respectivos anexos.

CAPÍTULO I: EL PROBLEMA DE INVESTIGACIÓN

1.1. Descripción del problema

El clima es un factor importante para la producción agrícola, sin embargo, durante las épocas de heladas (descenso de la temperatura al punto de congelación del agua que se da mayormente a alturas 3000 msnm) los cultivos se ven afectados drásticamente y sus efectos son devastadores para la economía de los agricultores.

El fenómeno de la helada es impredecible haciendo que los cultivos se vean afectados en momentos inesperados y la gran mayoría de agricultores no cuentan con métodos confiables y eficaces para combatir este fenómeno, los métodos comúnmente utilizados para ayudar a reducir los efectos de las heladas son costosos y poco eficaces.

En el Perú, en las zonas alto andinas de Puno, los campesinos utilizan técnicas artesanales como la quema de hierbas con la finalidad de aumentar la temperatura en las zonas de cultivo para reducir el efecto de las heladas, los cuales no son eficientes dado que gran parte del calor generado sube rápidamente y se pierde.

La eficacia de algunos métodos como el uso de calentadores de diésel y calefactores no siempre son confiables, en el caso de los calefactores se necesita un gran número de estos para poder cubrir todo el terreno por lo cual representa un gran gasto económico. Y por el lado de calentadores, el calor que emiten se dispersa fácilmente con la presencia de viento.

La contaminación que se produce con los métodos convencionales es muy perjudicial para el medio ambiente puesto que se tiene que quemar combustibles fósiles.

Asimismo, los agricultores no cuentan con sistemas de monitoreo en tiempo real que les permitan anticipar la ocurrencia de heladas y activar mecanismos de protección de forma oportuna y automática. Ante esta limitación, la incorporación de tecnologías basadas en IoT se presenta como una alternativa viable y adecuada para las condiciones agroclimáticas de las zonas altoandinas. El registro continuo de variables ambientales como temperatura, humedad relativa, velocidad del viento y radiación térmica que posibilita la identificación temprana de condiciones críticas asociadas a la formación de heladas. En este sentido, la implementación de un sistema automatizado capaz de activar el riego por aspersión únicamente cuando

exista riesgo de helada contribuye a la protección de los cultivos, promueve el uso eficiente del recurso hídrico y disminuye la dependencia de métodos tradicionales que suelen resultar poco efectivos y con impactos ambientales negativos.

1.2. Formulación del problema

1.2.1. Problema general

¿En qué medida el uso de riego automatizado reducirá los efectos de las heladas en las zonas alto andinas?

1.2.2. Problemas específicos

a. ¿Cuáles son los efectos que producen las heladas en los cultivos?
b. ¿Cuáles son los parámetros a controlar en los cultivos como producto de las heladas para el diseño de prototipo?
c. ¿Qué tecnología de IOT es la más adecuada para el monitoreo remoto del prototipo?

1.3. Justificación e Importancia

La finalidad de esta tesis es poder reducir los efectos de las heladas en los cultivos en zonas alto andinas del Perú mediante un sistema automático de riego por aspersión usando microcontroladores ESP32 y Raspberry pi para poder monitorear local y remotamente los parámetros usando IoT.

Las heladas son uno de los fenómenos climáticos más devastadores para la agricultura, ya que pueden destruir cultivos enteros en cuestión de horas, generando pérdidas millonarias para los agricultores. Desde una perspectiva económica un sistema de riego por aspersión automatizado y monitoreado en tiempo real permite proteger los cultivos al formar una capa de hielo que actúa como aislante térmico, evitando que las plantas sufran daños. Esto significa en una considerable reducción de pérdidas económicas, ya que no será necesario replantar cultivos ni compensar ingresos perdidos. Además, el uso de sensores y tecnologías IoT genera el aprovechamiento de agua y energía, lo que reduce los costos operativos a largo plazo. Por último, al garantizar cultivos de mejor calidad y rendimiento, los agricultores pueden acceder a

mercados más competitivos, obteniendo precios más favorables y aumentando sus ingresos.

Desde una perspectiva social, este sistema trae beneficios significativos para las comunidades agrícolas. Al proteger los cultivos de las heladas, ayuda a garantizar la seguridad alimentaria, asegurando un suministro constante de alimentos, algo clave en regiones donde la agricultura es el principal sustento. También mejora la calidad de vida de los agricultores, ya que reduce la incertidumbre y el estrés que provocan las pérdidas de cosechas. Al automatizar el riego, disminuye la necesidad de trabajo manual, permitiendo que los productores puedan enfocarse en otras tareas, ya sean productivas o personales. Otro punto importante es que impulsa el desarrollo de habilidades tecnológicas en zonas rurales. La implementación de sensores, microcontroladores y plataformas IoT no solo moderniza el sector agrícola, sino que también capacita a los agricultores en el uso de herramientas digitales, abriéndose nuevas oportunidades tanto laborales como económicas.

Desde una perspectiva tecnológica, la automatización del riego mediante sensores ambientales permite monitorear las condiciones del entorno en tiempo real y tomar decisiones precisas, activando el riego solo cuando es estrictamente necesario. Esto no solo mejora la eficiencia del sistema, sino que también reduce el margen de error humano. El uso de microcontroladores ESP32 y Raspberry Pi proporciona una base tecnológica robusta y escalable, capaz de adaptarse a diferentes tipos de cultivos y extensiones de terreno. Además, la integración con plataformas IoT como Adafruit IO permite el monitoreo remoto, lo que facilita la supervisión y el control del sistema desde cualquier lugar, a través de dispositivos móviles. Esto es especialmente útil para agricultores que gestionan grandes extensiones de tierra o que no pueden estar físicamente presentes en el campo en todo momento. La capacidad de almacenar y analizar datos pasados es otra ventaja clave, ya que permite identificar patrones climáticos, mejorar la eficiencia del sistema y planificar estrategias a largo plazo.

1.4. Objetivos

1.4.1. Objetivo general

Diseñar e implementar un prototipo de monitoreo de riego por aspersión que reduzca los efectos de las heladas en los cultivos de las zonas alto andinas del Perú.

1.4.2. Objetivos específicos

a. Determinar cuáles son los efectos de las temperaturas bajas producto de las heladas en los cultivos en las zonas alto andinas
b. Determinar los parámetros a controlar en los cultivos como producto de las heladas y los parámetros de hardware y software del prototipo
c. Determinar qué tecnología IoT es las más adecuada para el monitoreo remoto del prototipo

CAPÍTULO II: MARCO TEÓRICO

2.1. Antecedentes de la investigación

Para dar una amplia recopilación de antecedentes se revisaron varias tesis de repositorios de universidades nacionales e internacionales.

2.1.1. Antecedentes internacionales

Arhancet (2024), en su tesis “Evaluación de tres sistemas de control de heladas mediante riego por aspersión en cultivo de cerezos (Prunus avium L.) en el Valle de Los Antiguos, provincia de Santa Cruz” cuyo objetivo fue evaluar tres sistemas de control de heladas mediante la implementación de riego por aspersión convencional, microaspersión subarbóreo y de baja presión, obtuvo como resultado que el sistema de riego por aspersión convencional fue el más eficaz ante heladas moderadas y fuertes.

Cevallos y Mosquera (2022), en su trabajo de titulación “Diseño e implementación de un prototipo IoT para el monitoreo de parámetros ambientales aplicados al cultivo de arroz utilizando ESP32 y ThingSpeak” se propusieron como objetivo principal desarrollar e implementar un prototipo IoT que empleara microcontroladores ESP32 y la plataforma ThingSpeak, destinado al monitoreo de los parámetros ambientales de un cultivo de arroz. Como resultado, lograron con éxito la recolección, almacenamiento y análisis de datos, así como el control del prototipo mediante la plataforma ThingSpeak.

Proaño (2024), en su trabajo de integración curricular “Implementación de un prototipo de un sistema de monitoreo de cantidad de luz, humedad del suelo y temperatura, para cultivo de plantas, basado en esp32 y página web” implementó un prototipo de monitoreo remoto basado en ESP32 y una página web (PHP y HTML) para la recolección de parámetros ambientales importantes para el cultivo de plantas. Obtuvo resultados favorables en cuanto al almacenamiento y recolección de dichos parámetros.

2.1.2. Antecedentes nacionales

Huanca (2024), en su tesis Implementación de un sistema inalámbrico utilizando el internet de las cosas para el monitoreo del cultivo de papas en el fundo Huaylacucho en el departamento de Ayacucho, cuyo objetivo fue implementar un sistema integrado de

monitoreo para el cultivo de papas, utilizó plataformas como Telegram, ThingSpeak y Google Sheets, junto al microcontrolador ESP32, obteniendo resultados positivos luego de haber implementado el sistema en dicho cultivo.

Castro (2023), en su trabajo de titulación “Smart Greenhouse aplicando las IoT para la conservación del cultivo de tomates nativos en la región Lambayeque” implementó un sistema de IoT con ayuda de un microcontrolador ESP32 para el monitoreo y la conservación de un ambiente óptimo para los cultivos de tomates peruanos. El proyecto se implementó en un Greenhouse con cinco variedades de tomates, logrando un rendimiento mayor en un 4,17%.

Rubiños (2024), en su informe final de proyecto de investigación “Monitoreo de la calidad del agua mediante tecnología lora en un sistema hidropónico para mejorar el cultivo de lechuga en zonas urbanas de Ventanilla, Callao 2024” cuyo objetivo fue diseñar e implementar un sistema para la adquisición y registro de datos para el seguimiento de las condiciones de temperatura, pH, TDS y conductividad eléctrica de cultivos hidropónicos de lechuga, empleó el microcontrolador ESP32 para enviar los datos a la plataforma Firebase para su posterior procesamiento. Los resultados obtenidos permitieron realizar los ajustes correspondientes para que el sistema funcionara de manera óptima, logrando una mejora en el crecimiento de las plantas.

2.2. Bases teóricas

2.2.1. Las Heladas

Las heladas son fenómenos meteorológicos que ocurren cuando la temperatura del aire cerca de la superficie terrestre desciende por debajo del punto de congelación del agua (0 °C). Este descenso de temperatura puede provocar la formación de hielo en las superficies expuestas, como plantas, suelos y estructuras (Organización Meteorológica Mundial [OMM], 1992, p. 256).

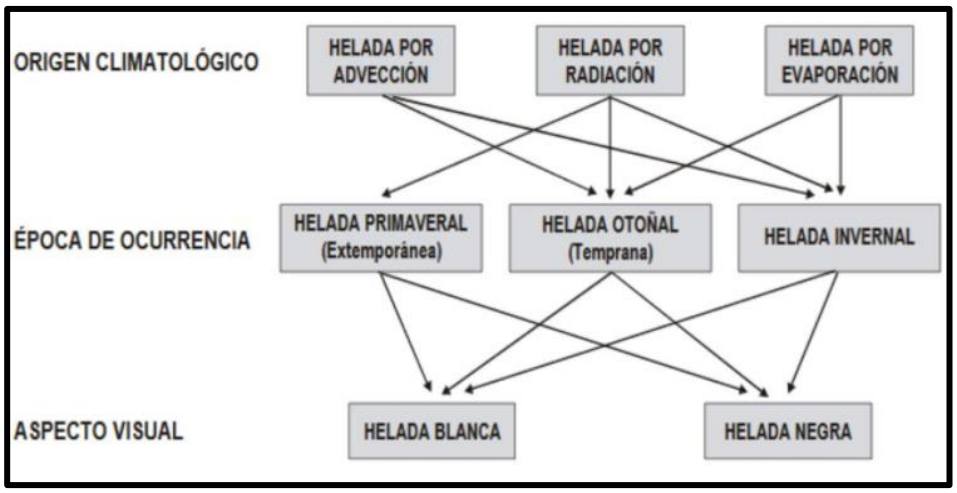
2.2.1.1. Clasificación

Según Matías (2007, como se citó en SENAMHI, 2010, p. 11), las heladas pueden clasificarse según su origen en advección, radiación y evaporación. En función de la época en que ocurren, se dividen en primaverales, otoñales e invernales. Además, según su apariencia visual, se distinguen en heladas blancas y negras.

En la Figura 1, se puede apreciar los tipos de heladas respecto a su origen, época y aspecto visual.

Figura 1

Tipos de heladas



Nota. Adaptado de *Atlas de las heladas del Perú* (Matías, 2007, como se cita en SENAMHI, 2010, p. 11).

- *Helada por Advección:* Las heladas de advección ocurren debido al desplazamiento de grandes masas de aire frío provenientes de las regiones polares, lo que genera diversas condiciones atmosféricas según el relieve del terreno. Este tipo de heladas suele presentarse en zonas bajas de montañas, como valles y cañadas, donde pueden causar daños significativos (Pereyra, 1990, como se citó en SENAMHI, 2010).

- *Helada de radiación:* Ocurren durante noches despejadas, cuando la superficie terrestre emite radiación hacia la atmósfera debido a la ausencia de nubosidad y a la baja concentración de vapor de agua. Además, suelen presentarse en condiciones de calma, es decir, sin la presencia de viento (Pereyra, 1990, como se citó en SENAMHI, 2010, p. 12). Este tipo de heladas es más frecuente en regiones de latitudes medias, especialmente a finales de la primavera y comienzos del otoño, siendo las principales responsables de las pérdidas en los cultivos. No obstante, también pueden presentarse durante el verano en áreas tropicales y subtropicales de mayor altitud (Elías & Castellví, 2001, como se citó en SENAMHI, 2010, p. 12).

- *Helada de evaporación:* Se produce cuando el agua acumulada sobre las plantas se evapora, lo que provoca un enfriamiento debido a la absorción del calor latente del aire. Si, tras una precipitación, la humedad relativa del aire disminuye, como suele suceder después del paso de un frente frío, el agua que cubre la vegetación se evapora rápidamente. La intensidad de estas heladas depende de la cantidad de agua evaporada, así como de la temperatura del aire y la humedad relativa (Elías & Castellví, 2001, como se citó en SENAMHI, 2010, p. 13).

2.2.1.2. Temperaturas críticas de la papa

La papa es un cultivo altamente sensible a las bajas temperaturas. Según la FAO (2010), presenta umbrales críticos que varían según su fase fenológica: en germinación y fructificación, sufre daños entre -2 °C y -3 °C, mientras que, en floración, los daños pueden ocurrir desde -1 °C. Estas temperaturas mínimas marcan el límite a partir del cual pueden producirse efectos como necrosis, reducción del rendimiento o pérdida del cultivo. Por ello, son un parámetro clave en el diseño de sistemas de protección contra heladas.

En la tabla 1, se puede apreciar las temperaturas críticas de diferentes tipos de cultivos según su fase de desarrollo.

Tabla 1

Temperaturas críticas por cultivo según etapa fenológica

Cultivo	Germinación (°C)	Floración (°C)	Fructificación (°C)
Trigo de primavera	-9 a -10	-1 a -2	-2 a -4
Avenas	-8 a -9	-1 a -2	-2 a -4
Cebada	-7 a -8	-1 a -2	-2 a -4
Guisantes	-7 a -8	-2 a -3	-3 a -4
Lentejas	-7 a -8	-2 a -3	-2 a -4
Almorta	-7 a -8	-2 a -3	-2 a -4
Cilantro	-8 a -10	-2 a -3	-3 a -4
Amapolas	-7 a -10	-2 a -3	-2 a -3
Diente de león	-8 a -10	-3 a -4	-3 a -4
Altramuz	-6 a -8	-3 a -4	-3 a -4
Veza de primavera	-6 a -7	-3 a -4	-2 a -4

Tabla 2 (Continuacion)

Cultivo	Germinación (°C)	Floración (°C)	Fructificación (°C)
Judías	-5 a -6	-2 a -3	-3 a -4
Girasol	-5 a -6	-2 a -3	-2 a -3
Cártamo	-4 a -6	-2 a -3	-3 a -4
Mostaza blanca	-4 a -6	-2 a -3	-3 a -4
Lino	-5 a -7	-2 a -3	-2 a -4
Cáñamo	-5 a -7	-2 a -3	-2 a -4
Remolacha azucarera	-6 a -7	-2 a -3	-
Zanahoria	-6 a -7	-	-
Col	-5 a -7	-2 a -3	-6 a -9
Soja	-3 a -4	-2 a -3	-2 a -3
Maíz	-2 a -3	-1 a -2	-2 a -3
Mijo	-2 a -3	-1 a -2	-2 a -3
Sorgo	-2 a -3	-1 a -2	-2 a -3
Patatas	-2 a -3	-1 a -2	-1 a -2
Algodón	-1 a -2	-1 a -2	-2 a -3
Melón	-0.5 a -1	-0.5 a -1	-1

*Nota***.** Adaptado de *Manual de prácticas agrícolas para el manejo de la sequía y otros factores climáticos* (p. 88), por FAO (2010).

2.2.1.3. Efectos de las heladas en los cultivos

Desde un punto de vista biológico, una helada ocurre cuando se forman cristales de hielo en las superficies debido a la congelación del rocío o por la transformación directa del vapor de agua en hielo. El estrés causado por las heladas abarca desde temperaturas frías inferiores a 20°C hasta temperaturas bajo cero, lo que impacta negativamente el crecimiento y desarrollo de las plantas, reduciendo la productividad agrícola (Espinoza, 2017).

- *Formación de hielo intracelular:* El primero tiene lugar cuando se produce la formación de cristales de hielo en el protoplasma celular, un proceso conocido como congelación intracelular, que puede comprometer la integridad de las células. Por otro lado, el daño indirecto se manifiesta cuando el hielo se genera dentro de los tejidos de la planta, pero permanece en el espacio extracelular, lo que puede provocar deshidratación celular y afectar la funcionalidad de los tejidos (FAO, 2010, p. 74).

- *Deshidratación celular:* La formación de hielo extracelular provoca un estrés hídrico secundario en las células adyacentes, ya que el agua se evapora desde el interior de las células y se deposita sobre los cristales de hielo fuera de ellas. A medida que el hielo continúa creciendo, las células se desecan más, lo que puede llevar al colapso celular debido a la desecación (Levitt, 1980, como se citó en FAO, 2010, p. 75).
- *Daño a los tejidos vegetativos:* El daño causado por las bajas temperaturas en las hojas se manifiesta principalmente por la pérdida de turgencia y la aparición de una tonalidad oscura en los tejidos, resultado de la acumulación de agua en los espacios intercelulares (Espinoza, 2017).

En la Figura 2, se puede apreciar los efectos de las heladas en un cultivo de papas.

Figura 2

Cultivos de papa afectado por heladas

*Nota***.** Adaptado de *Heladas afectan cultivos de papa* [Fotografía], por Radio Exa Asillo (s.f.), en https://www.facebook.com/share/14RkT6kZMs/

2.2.2. Parámetros a controlar en cultivos afectados por heladas

FAO (2010) nos dice que, para comprender y mitigar los efectos de las heladas en la agricultura, es fundamental tener en cuenta una serie de parámetros ambientales y agronómicos que influyen en la formación, intensidad y duración de este fenómeno.

2.2.2.1. Humedad relativa

La humedad relativa constituye un factor clave en la formación de heladas, debido a que un mayor contenido de vapor de agua en el aire favorece la condensación al

alcanzarse el punto de rocío; cuando las temperaturas descienden por debajo de 0 °C, dicha condensación se congela sobre las superficies, originando escarcha (Andrades Rodríguez & Muñez León, 2012).

2.2.2.2. Viento

El viento es un factor determinante en la ocurrencia de heladas, ya que las heladas de radiación se presentan generalmente en condiciones de calma atmosférica o con vientos muy débiles, lo que favorece el enfriamiento del aire cercano a la superficie y la formación de inversiones térmicas (FAO, 2010).

2.2.2.3. Tipo de suelo

El contenido de humedad del suelo influye en la ocurrencia de heladas, ya que los suelos secos presentan mayor cantidad de espacios de aire, lo que limita la transferencia y el almacenamiento de calor. En este sentido, la humedad del suelo favorece la retención de calor y mejora la protección contra heladas (FAO, 2010).

2.2.2.4. Temperatura crítica de los cultivos

La temperatura crítica de los cultivos corresponde al umbral térmico a partir del cual se producen daños en los tejidos vegetales, el cual varía según el tipo de cultivo y su etapa de desarrollo. Este parámetro resulta fundamental para la toma de decisiones en la activación de sistemas de protección contra heladas (FAO, 2010).

2.2.2.5. Radiación solar

La radiación solar influye en la formación de heladas, ya que, durante la noche, en condiciones de cielo despejado, se produce una pérdida de calor por radiación desde la superficie, lo que favorece el enfriamiento del aire cercano al suelo y la formación de inversiones térmicas (FAO, 2010).

2.2.2.6. Duración del riego

Durante la protección contra heladas mediante riego, la aplicación de agua debe mantenerse mientras persistan las condiciones de congelación, ya que su interrupción puede incrementar el daño en los cultivos (FAO, 2010).

2.2.2.7. Uniformidad de cobertura

La uniformidad de la aplicación del agua es fundamental en la protección contra heladas, ya que una distribución no homogénea puede dejar zonas del cultivo sin protección, incrementando el riesgo de daño por bajas temperaturas (FAO, 2010).

2.2.3. Riego por aspersión

El riego por aspersión es una técnica comúnmente empleada para proteger cultivos frente a las heladas, especialmente en regiones agrícolas con alta vulnerabilidad climática. Su funcionamiento se basa en el intercambio de calor sensible y latente: cuando el agua pulverizada entra en contacto con el ambiente frío o con las superficies vegetales, se congela y, al hacerlo, libera calor que ayuda a mantener la temperatura de los tejidos cercanos a los 0 °C. Esto previene que las plantas desciendan por debajo de su umbral de congelación, evitando así daños severos. No obstante, también se producen pérdidas de calor por evaporación y radiación, por lo que es esencial mantener una aplicación constante de agua durante todo el evento de helada. En algunos casos, como en los micro aspersores orientados al suelo, el objetivo es conservar el calor del sustrato, permitiendo que la planta reciba irradiación térmica desde la base. La efectividad de esta estrategia depende de variables como la temperatura inicial del agua, el caudal aplicado y las condiciones meteorológicas específicas del lugar (FAO, 2010).

La Figura 3 muestra un sistema de riego por aspersión en funcionamiento.

Figura 3

Sistema de riego por aspersión

Nota. Tomado de *Sistema de riego por aspersión: una protección ante las bajas temperaturas*, por Sertec Riego (s.f.), en https://sertecriego.com/sistema-de-riego-por-aspersion-una-proteccion-ante-las-bajas-temperaturas/

2.2.4. Tecnologías IOT para el monitoreo remoto del prototipo

2.2.4.1. Microcontrolador ESP32

El ESP32 es un microcontrolador de bajo costo con capacidades integradas de Wi-Fi y Bluetooth, desarrollado por Espressif Systems. Este chip se distingue por su capacidad de integrar soluciones de conectividad Wi-Fi en la banda de 2.4 GHz y Bluetooth 4.2 en un solo componente, lo que lo convierte en una solución altamente eficiente para aplicaciones en el ámbito del Internet de las Cosas (Kurniawan, 2019, p. 6).

En la Figura 4, se puede apreciar el microcontrolador ESP32.

Figura 4

Microcontrolador ESP32

*Nota***.** Adaptado de *Espressif Systems ESP32-DevKitC-32E* [Fotografía], por Mouser Electronics (s.f.), en https://www.mouser.pe/ProductDetail/Espressif-Systems/ESP32-DevKitC-32E?qs=GedFDFLaBXFpgD0kAZWDrQ%3D%3D

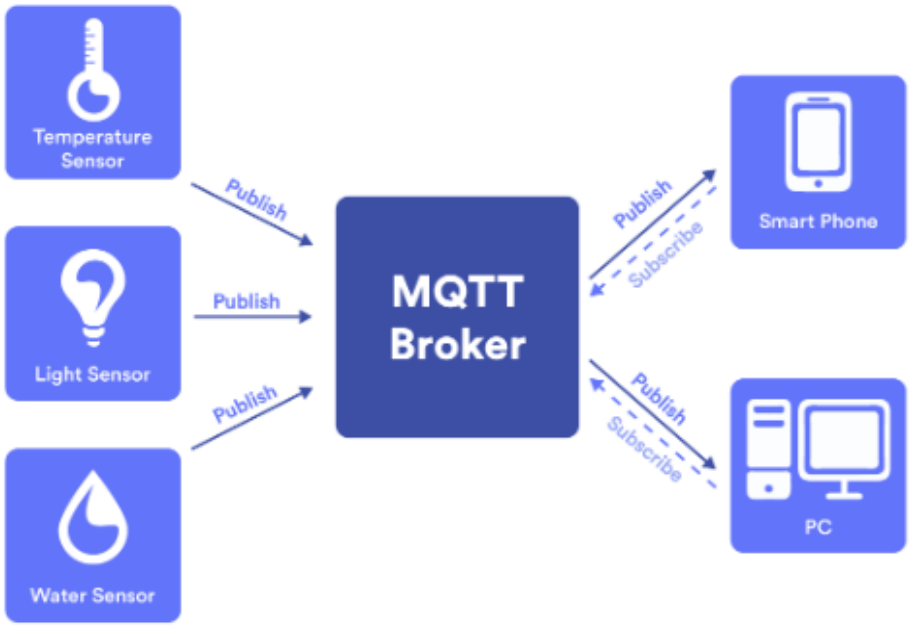
2.2.4.2. Protocolo MQTT

El protocolo MQTT (Message Queue Telemetry Transport) es un protocolo de mensajería asíncrona diseñado para facilitar las comunicaciones machine-to-machine en el ámbito del Internet de las Cosas (IoT). Su funcionamiento se basa en un modelo de publicación y suscripción, donde los dispositivos intercambian mensajes de manera eficiente. Este protocolo, conocido por su ligereza, es ideal para implementarse en redes con ancho de banda limitado y alta latencia. Además, su flexibilidad lo hace compatible con una amplia variedad de dispositivos y servicios IoT, lo que lo convierte en una opción versátil para aplicaciones en este campo (Mahedero, 2020).

La Figura 5 muestra el diagrama del funcionamiento del protocolo MQTT.

Figura 5

Diagrama del protocolo MQTT



*Nota***.** Tomado de *MQTT: El protocolo más adaptado para IoT* [Fotografía], por Pandora FMS (s.f.), en https://pandorafms.com/es/it-topics/que-es-mqtt/

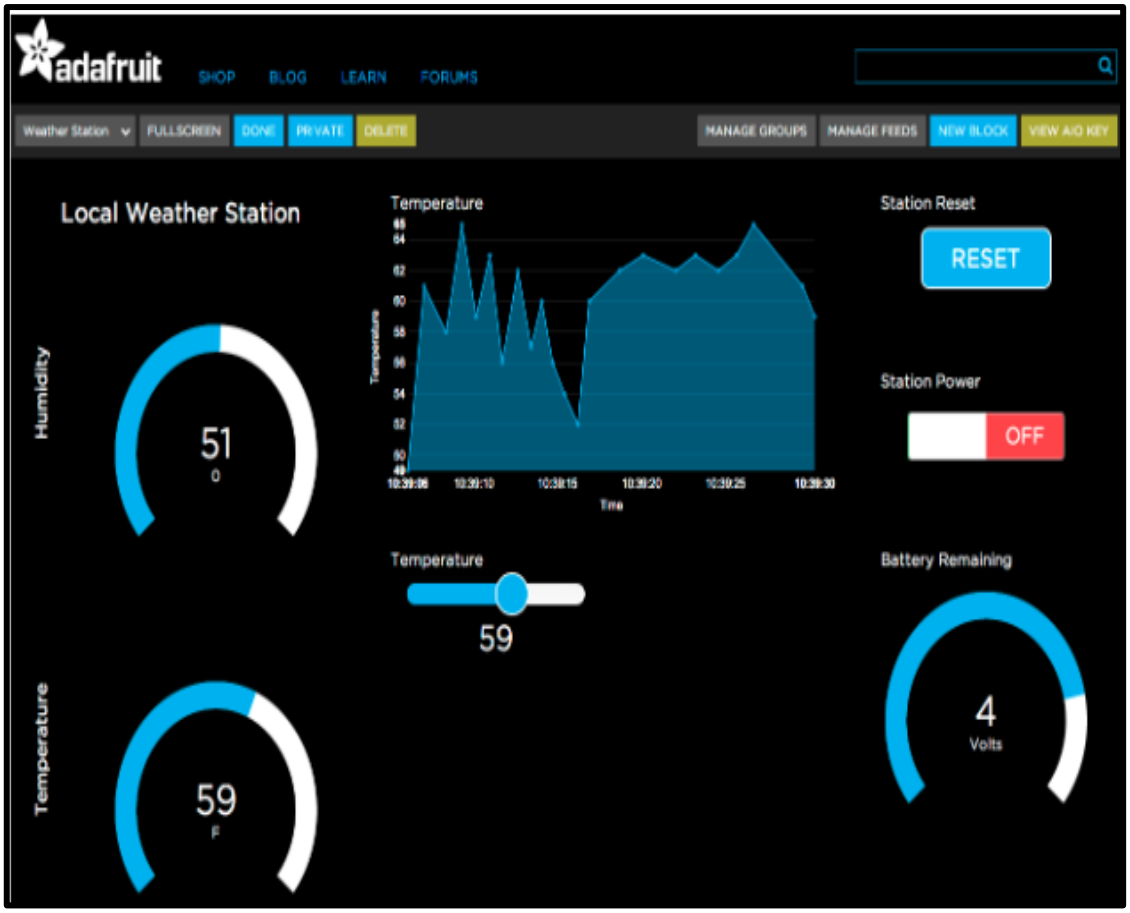
2.2.4.3. ADAFRUIT IO

Adafruit IO es una plataforma de Internet de las Cosas (IoT) desarrollada por AdafruitIndustries. La plataforma permite a los usuarios almacenar, visualizar e interactuar con datos en tiempo real a través de una interfaz web fácil de usar. Adafruit IO es compatible con una amplia gama de dispositivos, incluyendo Arduino, Raspberry Pi y otros. La plataforma ofrece tanto una API REST como soporte para MQTT, lo que permite una integración flexible y fácil para los desarrolladores. Además, Adafruit IO proporciona herramientas para crear dashboards personalizados, lo que permite a los usuarios monitorear y controlar sus proyectos IoT de manera visual y efectiva (Adafruit Industries, s.f.).

En la Figura 6, se muestra el interfaz del dashboard de Adafruit IO.

Figura 6

Interfaz de Adafruit IO



Nota. Adaptado de *Adafruit* [Fotografía], en https://io.adafruit.com

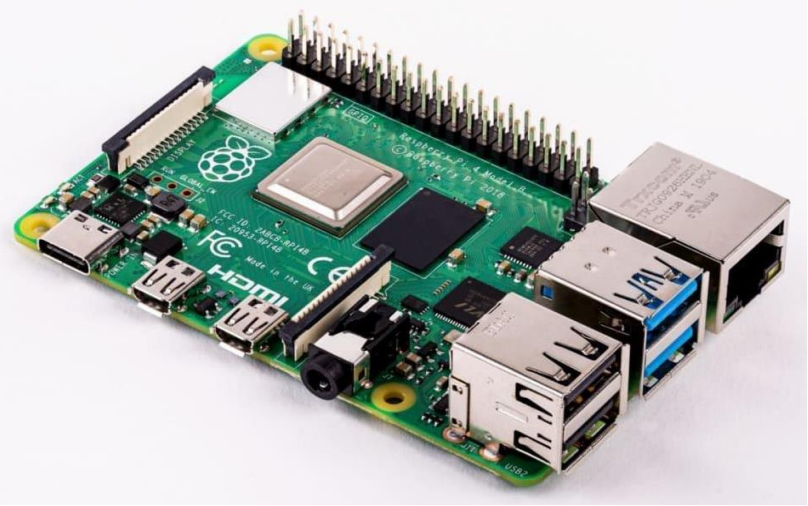
2.2.4.4. Raspberry Pi

Es una computadora de placa única del tamaño de una tarjeta de crédito, creada en el Reino Unido por Raspberry Pi Foundation con el propósito de fomentar la enseñanza de la informática en las escuelas. A pesar de que su principal objetivo era educativo, su bajo costo ha facilitado su uso en proyectos de hobby y sistemas embebidos. Ha sido empleada en diversas aplicaciones como la automatización, la seguridad doméstica y en el ámbito educativo en áreas como música y matemáticas. Desde su lanzamiento, ha transformado el desarrollo de sistemas embebidos, convirtiéndose en una plataforma accesible y económica para ingenieros y estudiantes (Halfacree, 2018, p. 8).

En la Figura 7, se puede apreciar el miniordenador Raspberry pi 3.

Figura 7

Raspberry Pi 3



Nota. Adaptado de *Raspberry: hay una computadora para todo* [Fotografía], por GoDaddy (s.f.), en https://www.godaddy.com/resources/es/tecnologia/que-es-raspberry-pi

2.2.4.5. Influx DB

Es una base de datos de series temporales diseñada para el almacenamiento rápido y de alta disponibilidad y la recuperación de datos de series temporales. Puede funcionar como una solución independiente, o se puede utilizar para procesar datos de Graphite. Además de la supervisión, InfluxDB se puede utilizar para el Internet de las cosas, los datos de sensores y las soluciones de automatización del hogar. (Stackhero, 2022).

Estas son algunas de las características que InfluxDB admite actualmente que lo convierten en una excelente opción para trabajar con datos de series temporales.

- Almacén de datos personalizado de alto rendimiento escrito específicamente para datos de series temporales.
- El motor TSM permite una alta velocidad de ingesta y compresión de datos
- API HTTP de escritura y consulta simples y de alto rendimiento.
- Los complementos son compatibles con otros protocolos de ingesta de datos como Graphite, collectd y OpenTSDB.
- Lenguaje de consulta expresivo similar a SQL adaptado para consultar fácilmente datos agregados.
- Las etiquetas permiten indexar series para consultas rápidas y eficientes.

- Las políticas de retención caducan automáticamente de manera eficiente los datos obsoletos.
- Las consultas continuas calculan automáticamente los datos agregados para que las consultas frecuentes sean más eficientes.

La Figura 8 muestra el interfaz de Influx DB.

Figura 8

Diagrama de Influx DB

Nota. Adaptado de *InfluxDB: Introducción* [Fotografía], Stackhero. https://www.stackhero.io/es-ES/services/InfluxDB/documentations/Introduccion

2.2.4.6. Telegram

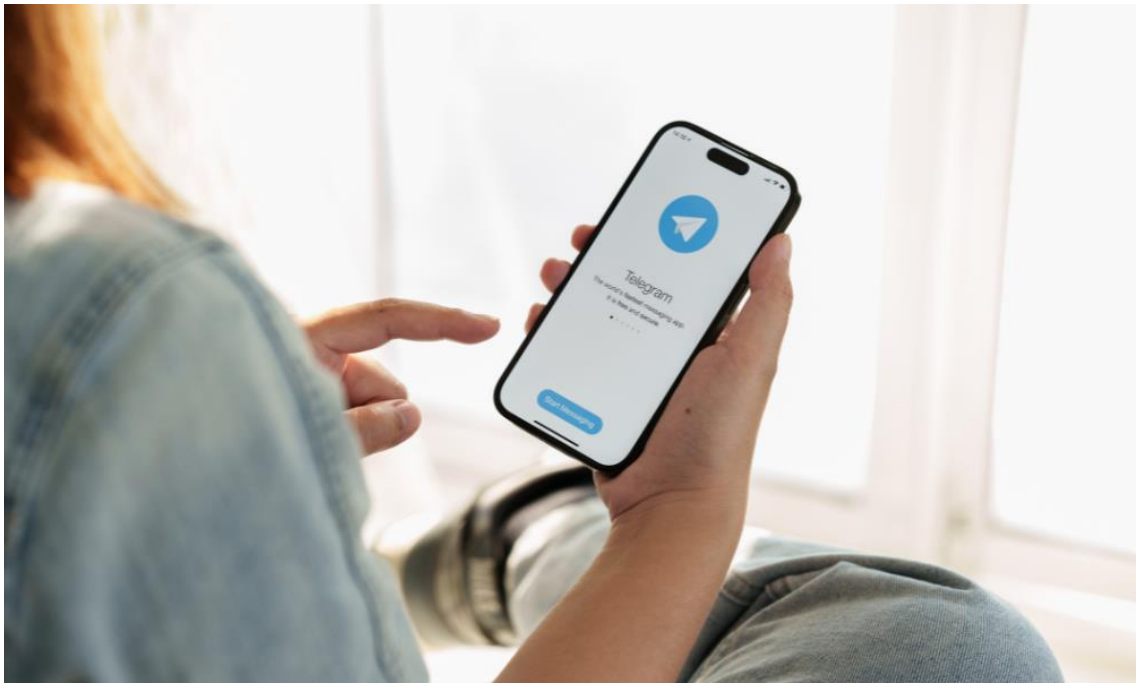
Telegram es una plataforma de mensajería instantánea que permite a los usuarios enviar mensajes de texto, compartir archivos multimedia, realizar llamadas de voz y vídeo y unirse a grupos y canales para recibir información.

Pone sus máximos esfuerzos en la seguridad y la velocidad, lo que la ha convertido en una de las alternativas más atractivas para quienes buscan una comunicación eficiente y, más importante aún, privada. (Go Daddy, 2024)

En la Figura 9, se puede observar el uso de la app Telegram en el día al día.

Figura 9

Persona usando app de mensajería Telegram



Nota. Adaptado de *Telegram: Qué es, cómo funciona y por qué usarlo como app de mensajería* [Fotografía], por GoDaddy (s.f.), en https://www.godaddy.com/resources/latam/general/que-es-telegram

2.3. Definición de términos

2.3.1. Microcontrolador

Dispositivo electrónico que ha sido diseñado para que se ejecuten procesos lógicos y poder realizar una tarea específica; dicha tarea es programada a través de un lenguaje de programación por el usuario (UNAM, s.f.).

2.3.2. Internet de las cosas

Son sistemas que constan de dos partes fundamentales: hardware y software; el hardware se encarga de recopilar datos y el software se encarga de guardar y enviar dichos datos a través de redes inalámbricas hasta un punto deseado, éstas dos partes fundamentales hacen que la intervención humana se reduzca al mínimo (Red Hat, 2023).

2.3.3. Riego por aspersión

Es un sistema de riego que aplica agua a los cultivos de manera constante y uniforme, imitando a la lluvia, con el riego por aspersión se hace un uso manejable del recurso hídrico lo que genera mejor aprovechamiento del agua, reduciendo los costos entre 70 y 90% (Biblioteca Digital INIA, 2020).

2.3.4. Sensores temperatura

Son dispositivos encargados de convertir la señal de la temperatura analógica a una señal digital de manera precisa, son usados en aplicaciones industriales para controlar circuitos que dependen de la temperatura (Rechner, 2022).

CAPÍTULO III: MARCO METODOLÓGICO

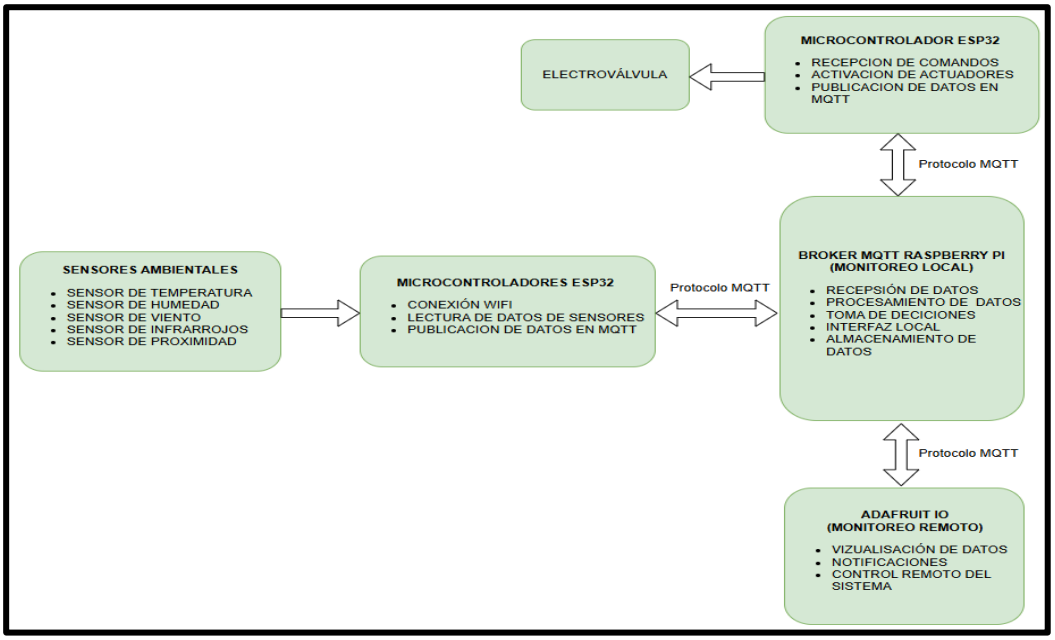
3.1. Diseño de la investigación

El presente proyecto se enmarca dentro de una investigación tipo experimental, ya que su desarrollo implica la creación, implementación y validación de un sistema basado en componentes específicos como el microcontrolador ESP32, sensores de temperatura, sensores de humedad, sensores de viento, sensores infrarrojos, sensores de proximidad y un sistema de riego por aspersión. Este enfoque experimental se justifica en la necesidad de integrar estos elementos en un prototipo funcional que permita monitorear y controlar de manera automatizada las condiciones ambientales en cultivos de zonas altoandinas, con el fin de mitigar los efectos de las heladas. A través de este proceso, se busca evaluar la eficacia del sistema en condiciones controladas, ajustando variables como el valor de los sensores, la frecuencia de riego, y la eficiencia de la transmisión de datos mediante el microcontrolador ESP32 y el miniordenador Raspberry Pi en tiempo real a través de plataformas IoT.

En la Figura 10, se puede apreciar el diagrama del bloque del prototipo.

Figura 10

Diagrama de bloques del prototipo



3.2. Acciones y actividades

3.2.1. Determinación de parámetros del prototipo

De acuerdo con la revisión de fuentes especializadas, como el Atlas de Heladas del Perú del SENAMHI (2010) y publicaciones de la FAO (Organización de las Naciones Unidas para la Alimentación y la Agricultura), se identifican los principales efectos de las heladas en los cultivos, los cuales son:

- Daño celular por congelamiento.
- Deshidratación.
- Reducción del rendimiento.
- Daño a los tejidos vegetativos.

En base a estos efectos, se determinaron los principales parámetros a controlar en el sistema de mitigación, entre los cuales se incluyen:

- Temperatura del aire.
- Humedad relativa.
- Humedad del suelo.

Adicionalmente, se consideraron otros parámetros complementarios para mejorar la efectividad del sistema, contribuyendo a la protección de los cultivos y evitando la disminución de su rendimiento:

- Velocidad del viento km.
- Nivel de agua disponible para el riego de cultivos.
- Temperatura de la atmosfera para determinar si el cielo está despejado o nublado lo cual es importante para la detección anticipada de heladas por radiación.

Una vez identificado los parámetros a tener en cuenta se procedió con la elección de los distintos sensores que ayudaran a mitigar los efectos de las heladas en los cultivos, los cuales fueron los siguientes:

- Sensor de Temperatura -20°C a 100 °C
- Sensor 0-100% RH
- Sensor de Humedad de suelo
- Anemómetro de 3 tasas 0 a 100 km/h
- Sensor ultrasónico 0 a 4m
- Sensor de temperatura infrarrojo -20 °C a 100 °C

3.2.2. Determinación de sensores del prototipo

Una vez identificados los parámetros críticos a monitorear, se procedió con la selección de los sensores adecuados para mitigar los efectos de las heladas en los cultivos. Los sensores elegidos permiten detectar condiciones de riesgo y activar el sistema de riego por aspersión de forma eficiente. A continuación, se detallan los dispositivos seleccionados:

- *Sensor de temperatura DS18B20:* Rango de medición de -55 °C a 125 °C; utilizado para monitorear la temperatura del aire con alta precisión.
- *Sensor combinado DHT22:* Capaz de medir humedad relativa (0–100 % RH) y temperatura; útil para evaluar el microclima alrededor del cultivo.
- *Sensor de humedad de suelo capacitivo v1.2:* Permite determinar el contenido de humedad en el suelo, lo que ayuda a evitar el estrés hídrico o el riego innecesario.
- *Anemómetro de tres tazas:* Mide la velocidad del viento en un rango de 0 a 100 km/h; importante para evitar activar el sistema cuando hay viento fuerte que podría dispersar el agua.
- *Sensor ultrasónico HC-SR04:* Mide la distancia del nivel de agua en el reservorio (0 a 4 m); garantiza que haya suficiente agua disponible antes de activar el sistema.
- *Sensor de temperatura infrarrojo MLX90614:* Con rango de -40 °C a 170 °C, permite medir la temperatura del cielo o de superficies sin contacto; útil para detectar heladas por radiación.

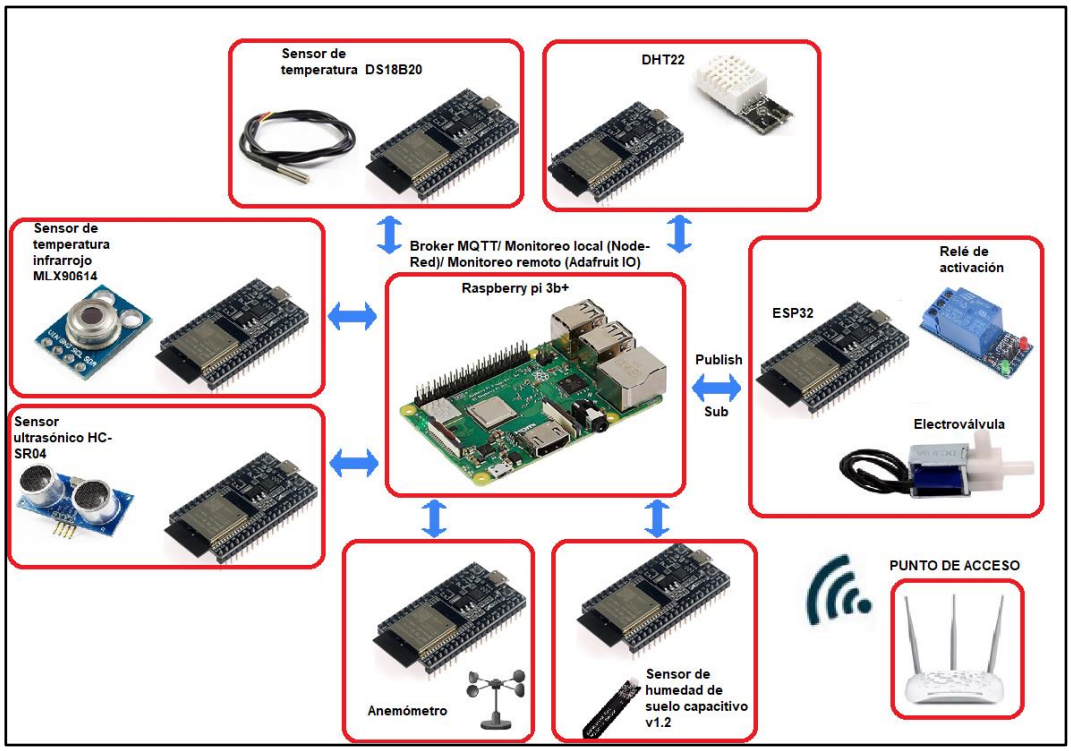
3.2.3. Diseño de hardware

El presente proyecto tiene como objetivo el diseño de un prototipo de sistema de riego por aspersión automatizado, orientado a reducir el impacto de las heladas en cultivos agrícolas. El hardware está conformado por sensores ambientales, microcontroladores y actuadores.

La Figura 11 muestra el diagrama general de hardware del prototipo.

Figura 11

Diagrama general del hardware



El sistema diseñado tiene como finalidad activar de manera automática un mecanismo de riego por aspersión cuando se detectan condiciones ambientales asociadas a riesgo de heladas. Para ello, se implementó una red de sensores conectados a microcontroladores ESP32, encargados de medir parámetros críticos como temperaturas, humedad relativa, velocidad del viento, humedad del suelo y nivel de agua. Cada ESP32 procesa localmente la información de su sensor y la transmite mediante el protocolo MQTT a un bróker alojado en una Raspberry Pi 3.

La Raspberry Pi actúa como nodo central del sistema, gestionando los mensajes recibidos y publicándolos en un dashboard desarrollado en Node-RED para visualización local en tiempo real, simultáneamente los datos se envían a la plataforma Adafruit IO para su almacenamiento y monitoreo remoto. Cuando se detectan condiciones previamente definidas como críticas, se activa una válvula solenoide controlada desde la Raspberry Pi, iniciando el riego por aspersión en la zona afectada. Cuando la temperatura se estabiliza o si hay vientos fuertes el sistema detiene automáticamente el riego. Este funcionamiento busca responder de forma preventiva o reactiva ante eventos de helada, reduciendo el riesgo de daño a los cultivos

3.2.3.1. Selección de componentes

Para la selección de los componentes del prototipo, se consideraron aspectos como el consumo energético, la precisión, el costo y el rango de funcionamiento en condiciones de bajas temperaturas.

a. ESP32

El ESP32 fue seleccionado por su bajo consumo energético, capacidad de procesamiento en tiempo real, conectividad inalámbrica integrada y facilidad de programación mediante el entorno Arduino IDE. El ESP32 es responsable de leer los datos del sensor conectado, procesarlos localmente y transmitirlos mediante MQTT al bróker central en la Raspberry Pi.

b. Raspberry Pi 3 Model B+

Su versatilidad y compatibilidad con diversas plataformas de monitoreo lo hacen ideal como nodo central en el sistema. Actúa como bróker MQTT, servidor para Node-RED y controlador central del sistema, además de almacenar y redirigir datos hacia la nube (Adafruit IO).

c. Sensor de Temperatura DS18B20

El DS18B20 destaca por su precisión, facilidad de integración y durabilidad, especialmente en entornos húmedos o con múltiples puntos de medición. Su protocolo 1-Wire simplifica el diseño electrónico, y su encapsulado resistente lo hace ideal para aplicaciones en exteriores o condiciones adversas.

d. Sensor DHT22

Sensor ampliamente utilizado en prototipos IoT por su facilidad de uso y buena precisión para monitoreo ambiental.

e. Sensor de humedad de suelo capacitivo v1.2

A diferencia de los sensores resistivos, este sensor capacitivo no se corroe con el tiempo, lo que lo hace más duradero y confiable en campo.

f. Anemómetro de 3 cazoletas

Su señal de salida es fácilmente leída por microcontroladores como esp32 sin requerir una alimentación.

g. Sensor ultrasónico HC-SR04

Sensor económico y de fácil implementación. Útil para conocer el nivel de agua disponible antes de iniciar el riego.

h. Sensor de temperatura infrarrojo MLX90614

Ideal para detectar variaciones térmicas rápidas sin contacto físico, lo que permite identificar puntos fríos críticos antes de que ocurran daños por heladas.

i. Válvula solenoide 12 V DC

Se seleccionó por su compatibilidad con control mediante relés, buena respuesta de apertura/cierre y fiabilidad para automatización de riego. Se activa cuando el sistema detecta condiciones de riesgo permitiendo el paso de agua a través del sistema de riego por aspersión.

3.2.3.2. Diagrama de conexión de nodos sensores

- **Nodo Sensor - DHT22**

En la Tabla 2, se puede observar la asignación de pines según la programación del Nodo Sensor - DHT22.

Tabla 3

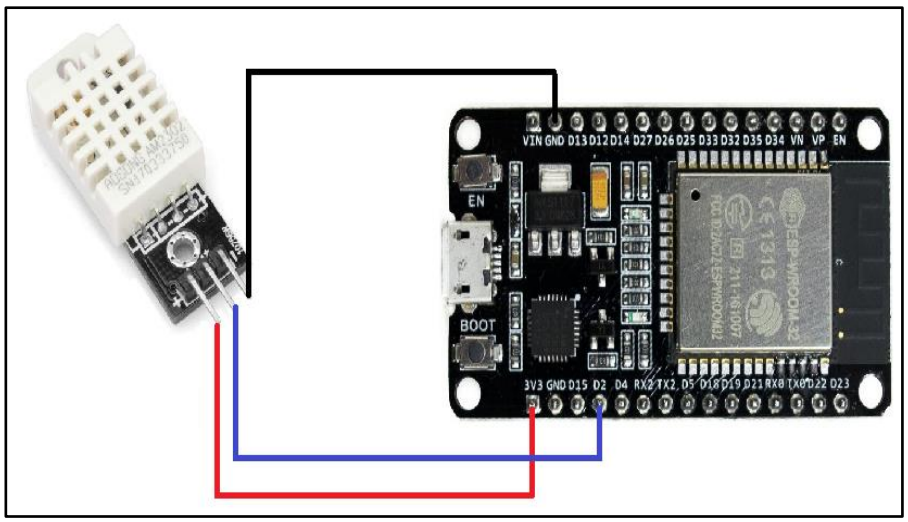
Pines de Conexión Eléctrica – Nodo Sensor – DHT22

Pin ESP32	Pin DHT22
Gpi04	Out
Gnd	Gnd
3v3	Vcc

En la Figura 12, se puede observar las conexiones eléctricas del microcontrolador ESP32 hacia el sensor DHT22, de acuerdo a lo que se observa en la Tabla 2.

Figura 12

Diagrama de Conexión Eléctrica - Nodo Sensor - DHT22



- **Nodo Sensor – DS18B20**

En la Tabla 3, se puede observar la asignación de pines según la programación del Nodo Sensor – DS18B20.

Tabla 4

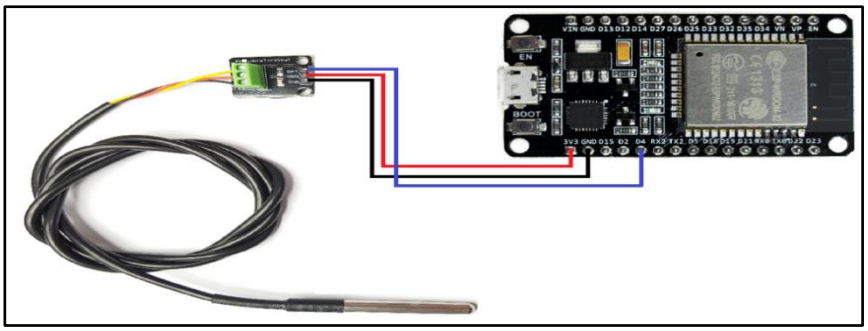
Pines de Conexión Eléctrica – Nodo Sensor – DS18B20

Pin ESP32	Pin DS18B20
Gpi04	Dat
Gnd	Gnd
3v3	Vcc

En la Figura 13, se puede observar las conexiones eléctricas del microcontrolador ESP32 hacia el sensor DS18B20, de acuerdo a lo que se observa en la Tabla 3.

Figura 13

Diagrama de Conexión Eléctrica - Nodo Sensor – DS18B20



- **Nodo Sensor - Anemómetro**

En la Tabla 4, se puede observar la asignación de pines según la programación del Nodo Sensor – Anemómetro.

Tabla 5

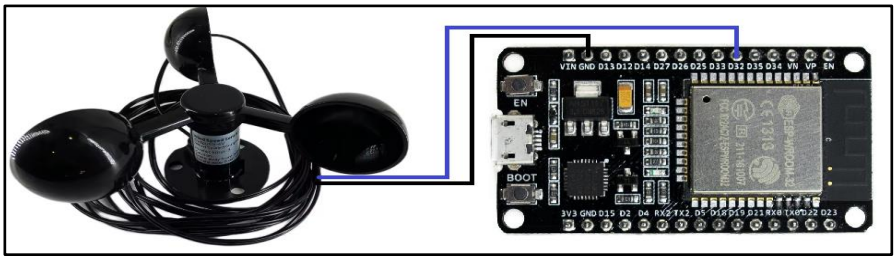
Pines de Conexión Eléctrica – Nodo Sensor – Anemómetro

Pin ESP32	Pin Anemómetro
Gpi032	Data
Gnd	Gnd

En la Figura 14, se puede observar las conexiones eléctricas del microcontrolador ESP32 hacia el sensor del anemómetro, de acuerdo a lo que se observa en la Tabla 4.

Figura 14

Diagrama de Conexión Eléctrica - Nodo Sensor – Anemómetro



- **Nodo Sensor – Humedad V1.2**

En la Tabla 5, se puede observar la asignación de pines según la programación del Nodo Sensor – Humedad.

Tabla 6

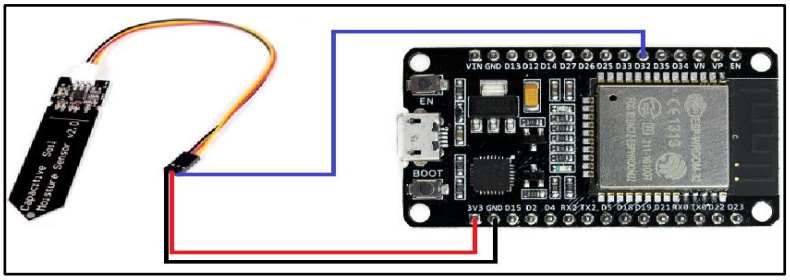
Pines de Conexión Eléctrica – Nodo Sensor – Humedad V1.2

Pin ESP32	Pin Humedad V1.2
Gpi032	Out
Gnd	Gnd
3v3	Vcc

En la Figura 15, se puede observar las conexiones eléctricas del microcontrolador ESP32 hacia el sensor de humedad V1.2, de acuerdo a lo que se observa en la Tabla 5.

Figura 15

Diagrama de Conexión Eléctrica - Nodo Sensor – Humedad V1.2



- **Nodo Sensor – Temperatura Infrarrojo MLX90614**

En la Tabla 6, se puede observar la asignación de pines según la programación del Nodo Sensor – Temperatura Infrarrojo MLX90614.

Tabla 7

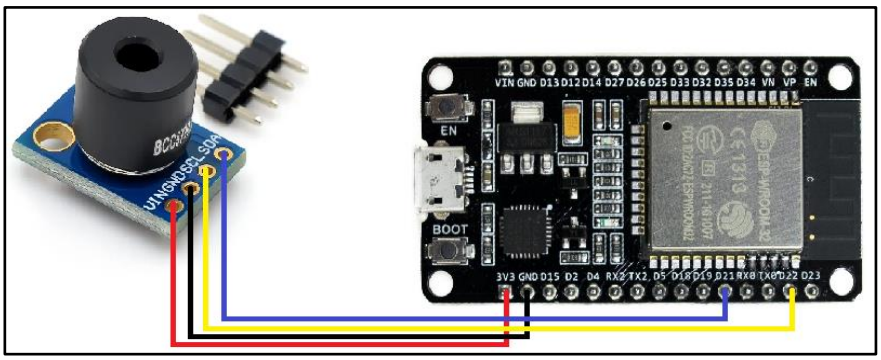
Pines de Conexión Eléctrica – Nodo Sensor – Temperatura Infrarrojo MLX90614

Pin ESP32	Pin MLX90614
Gpio21	Sda
Gpio22	Scl
3v3	Vin
Gnd	Gnd

En la Figura 16, se puede observar las conexiones eléctricas del microcontrolador ESP32 hacia el sensor de temperatura infrarrojo MLX90614, de acuerdo a lo que se observa en la Tabla 6.

Figura 16

Diagrama de Conexión Eléctrica - Nodo Sensor – Temperatura Infrarrojo MLX90614



- **Nodo Sensor – Ultrasónico HC-SR04**

En la Tabla 7, se puede observar la asignación de pines según la programación del Nodo Sensor – Ultrasónico HC-SR04.

Tabla 8

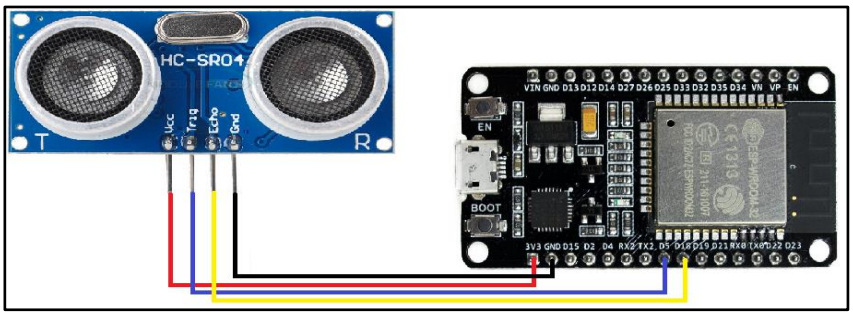
Pines de Conexión Eléctrica – Nodo Sensor – Ultrasónico HC-SR04

Pin ESP32	Pin HC-SR04
Gpio18	Echo
Gpio5	Trigger
5v	Vcc
Gnd	Gnd

En la Figura 17, se puede observar las conexiones eléctricas del microcontrolador ESP32 hacia el sensor ultrasónico HC-SR04 de acuerdo a lo que se observa en la Tabla 7.

Figura 17

Diagrama de Conexión Eléctrica - Nodo Sensor – Ultrasónico HC-SR04



- **Nodo Sensor - Relé**

En la Tabla 8, se puede observar la asignación de pines según la programación del Nodo actuador - relé.

Tabla 9

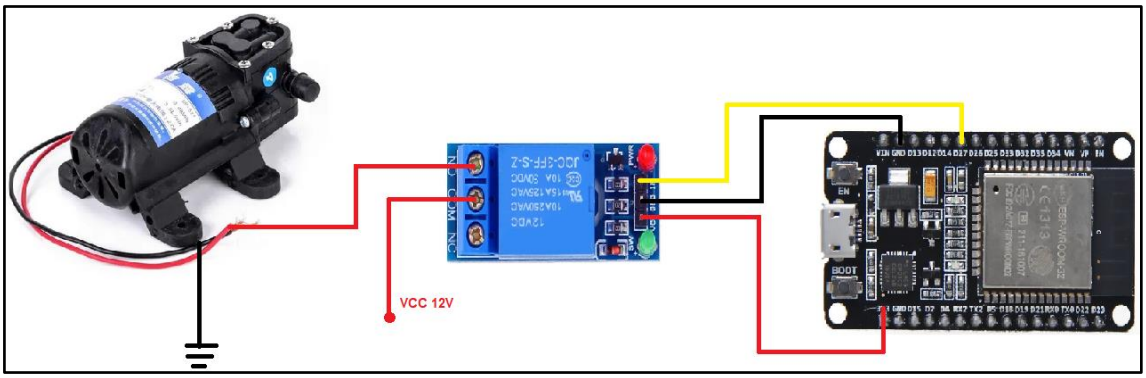
Pines de Conexión Eléctrica – Nodo actuador- Relé

Pin ESP32	Pin Relé
Gpi027	In
Gnd	Gnd
V5	Vcc

En la Figura 18, se puede observar las conexiones eléctricas del microcontrolador ESP32 hacia el Relé de 5V, de acuerdo a lo que se observa en la Tabla 8.

Figura 18

Diagrama de Conexión Eléctrica - Nodo actuador - Relé



3.2.3.3. Cálculo del consumo energético de los nodos ESP32

Según el documento técnico de Espressif (*ESP32 Power Consumption*, 2025), el consumo en modo activo con transmisión Wi-Fi alcanza valores promedio entre 160 y 260 mA, dependiendo de la potencia y actividad. En este trabajo, se asumió un consumo promedio de 200 mA para cada ESP32, considerando que están en uso continuo, recibiendo lecturas de sensores y transmitiendo mediante MQTT.

- **Corriente total para 6 nodos ESP32**

$$\boldsymbol{I_{total} = I * N}$$

$$I_{total} = 200\,mA * 6$$

$$I_{total} = 1200\,mA$$

- **Consumo energético diario (24 horas)**

  Para conocer el consumo energético diario, se multiplica la corriente total por el tiempo y por el voltaje de operación.

$$\boldsymbol{C_{diaria} = I_{total} * V * 24h}$$

$$C_{diaria} = 1.2A * 5V * 24h$$

$$C_{diaria} = 144\,Wh$$

El sistema, compuesto por 6 nodos ESP32 en operación continua, presenta un consumo energético diario de 144 Wh, por lo que para garantizar su funcionamiento ininterrumpido durante 24 horas se requiere una batería con una capacidad útil aproximada de 160 Wh, equivalente a unos 43 200 mAh a 3,7 V, considerando una eficiencia de descarga y conversión del 90 %.

3.2.4. Diseño de Software

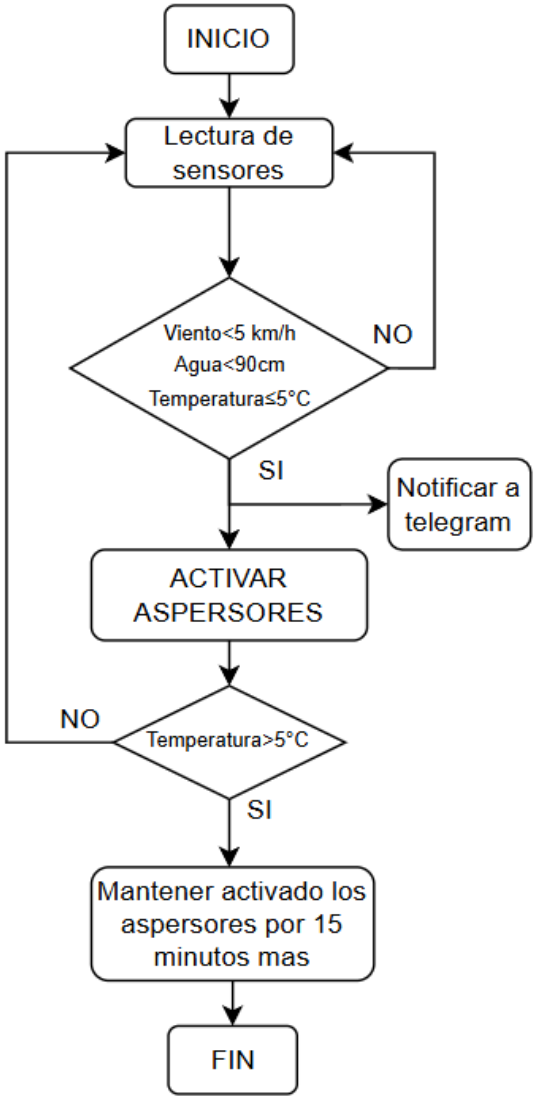
Para el desarrollo del presente proyecto se utilizó el entorno de programación Arduino IDE, destinado a la programación del microcontrolador ESP32, cuya función principal fue la lectura de datos provenientes de sensores ambientales y su posterior envío, mediante el protocolo MQTT, a un broker local basado en Mosquitto. El procesamiento de esta información, así como la toma de decisiones y la activación del sistema de riego, se llevaron a cabo en la plataforma Node-RED, donde también se implementó un panel de monitoreo local para la visualización en tiempo real. Además, los datos recolectados

se sincronizaron con la plataforma en la nube Adafruit IO, lo que permitió la supervisión remota y el almacenamiento histórico. También se integró un bot de Telegram en Node-RED, el cual permitió enviar notificaciones automáticas al usuario frente a eventos relevantes, tales como la detección de heladas o la presencia de valores críticos en los sensores.

En la Figura 19, se puede apreciar el diagrama de flujo de activación del prototipo ante heladas.

Figura 19

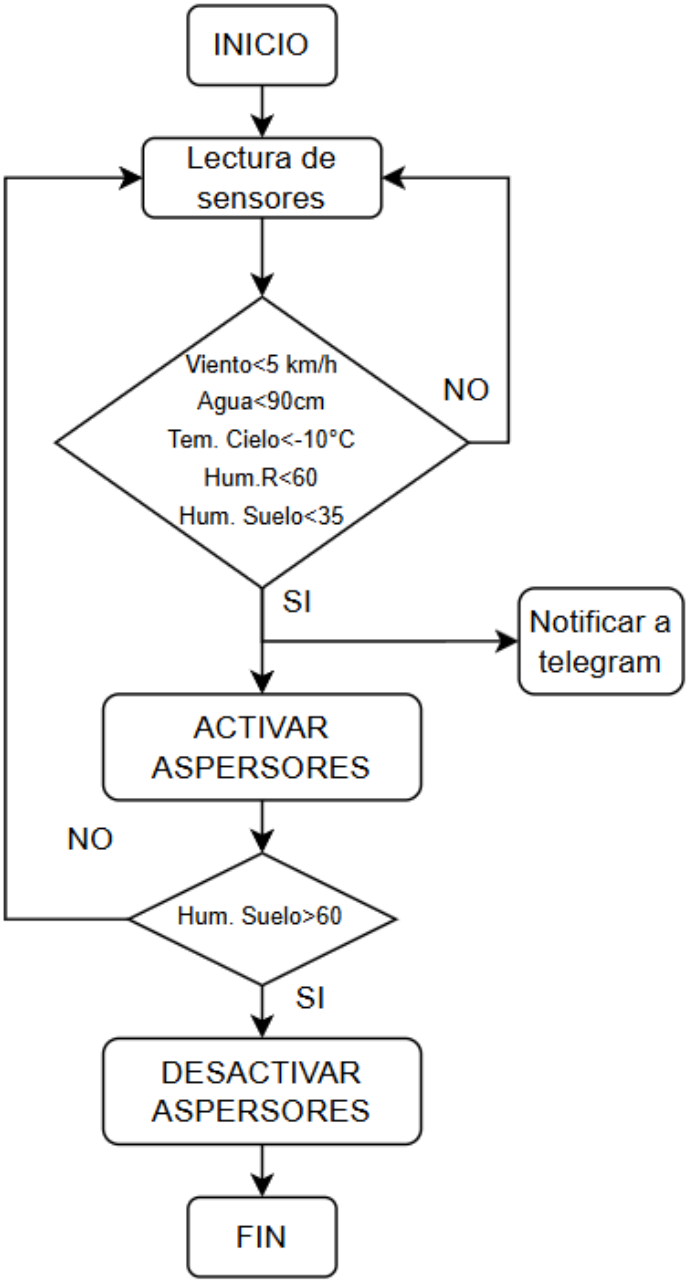
Diagrama de flujo de activación de aspersores ante la presencia de heladas



En la Figura 20, se puede apreciar el diagrama de flujo de activación preventiva ante posibles heladas.

Figura 20

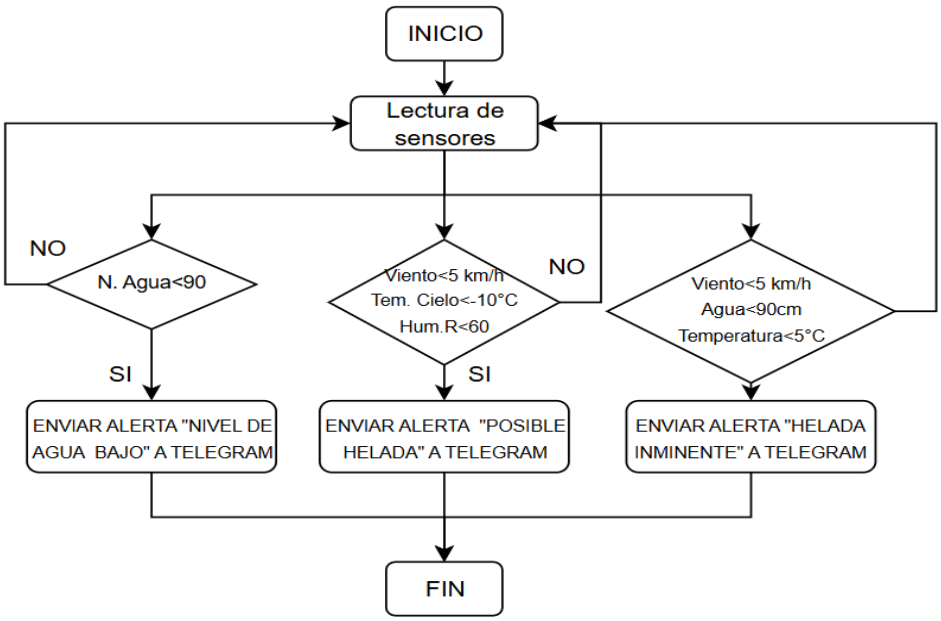
Diagrama de flujo de activación preventiva de aspersores ante posibles heladas



En la Figura 21, se puede apreciar el diagrama de flujo para el envío de notificaciones a Telegram.

Figura 21

Diagrama de flujo de envió de notificaciones a Telegram ante la presencia de heladas, posibles heladas y nivel de agua bajo



3.2.4.1. Softwares necesarios para el servidor MQTT

Para la implementación del servidor de monitoreo y control, se utilizaron diversos softwares fundamentales instalados en una Raspberry Pi 3 B+. Estos incluyen el sistema operativo Raspberry Pi OS, el broker MQTT **Eclipse Mosquitto**, la plataforma de visualización **Node-RED** y la base de datos **InfluxDB**, todos configurados para trabajar de forma conjunta en un entorno local.

Cada uno de estos componentes cumple una función específica:

- *Eclipse Mosquitto***:** actúa como el servidor MQTT para la comunicación entre los nodos ESP32.
- *Node-RED***:** permite visualizar datos y establecer reglas de actuación mediante flujos gráficos.
- *InfluxDB:* se encarga del almacenamiento de los datos recopilados por los sensores.

Los procedimientos detallados para la instalación y configuración de cada uno de estos softwares pueden consultarse en el Anexo 6.

3.2.4.2. Configuración de módulos ESP32 para la recolección de datos y activación de actuadores

Se definieron los parámetros de conexión a la red Wi-Fi y la dirección IP del broker MQTT. Además, se asignó un identificador único (clientID) a cada ESP32 para evitar conflictos entre dispositivos durante la conexión. Asimismo, se configuraron los tópicos de publicación y suscripción necesarios para la transmisión de datos.

En la Tabla 9, se utilizaron las siguientes librerías para la programación de los módulos ESP32.

Tabla 10
Librerías

Librería	Sensor	Función
#include <WiFi.h>	Todos	Permite al ESP32 conectarse a una red WiFi para transmitir datos.
#include <PubSubClient.h>	Todos	Maneja la comunicación con el broker MQTT (enviar/recibir datos).
#include <DHT.h>	DHT22	Permite leer temperatura y humedad del sensor DHT22.
#include <OneWire.h>	DS18B20	Habilita la comunicación 1-Wire del sensor DS18B20.
#include <DallasTemperature.h>	DS18B20	Librería de alto nivel para leer temperatura desde sensores DS18B20.
#include <Adafruit_MLX90614.h>	MLX90614	Permite leer temperatura del sensor infrarrojo MLX90614.
#include <Wire.h>	MLX90614	Habilita la comunicación I2C con el sensor MLX90614.

Nota. Elaboración con datos de la página de Arduino.

En la Tabla 10, se muestra los tópicos e identificadores asignados a los Esp32.

Tabla 11
Tópicos e id de los Esp32

Tópicos de Datos	Tópicos de Estados	ID ESP32
Sensor_DHT22	Estado/DHT22	DHT22_ESP32
Sensor_Humedad	Estado/Humedad	Humedad_ESP32
PIR	Estado/PIR	PIR_ESP32
Anemómetro	Estado/Anemómetro	Anemómetro_ESP32
Proximidad	Estado/Proximidad	Proximidad_ESP32
Temperatura	Estado/Temperatura	Temperatura_ESP32

En la Tabla 11, se puede apreciar las direcciones IP de los Esp32.

Tabla 12

IPS de los Esp32

Dirección IP	ID ESP32
192.168.1.17	DHT22_ESP32
192.168.1.19	Humedad_ESP32
192.168.1.99	PIR_ESP32
192.168.1.25	Anemómetro_ESP32
192.168.1.24	Proximidad_ESP32
192.168.1.23	Temperatura_ESP32

En la Figura 22, se muestra la asignación de las librerías necesarias, incluyendo las funciones de publicación y suscripción del protocolo MQTT, así como la configuración de las credenciales de conexión Wi-Fi, las direcciones IP, los tópicos y el identificador del cliente.

Figura 22

Asignación de librerías, credenciales wifi, ips, tópicos e ID

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>

#define DHTPIN 14
#define DHTTYPE DHT22

DHT dht(DHTPIN, DHTTYPE);

const int ledPin = 2;

// Datos WiFi
const char* ssid = "rolando 2.4G";
const char* password = "13121989";

// IP estática
IPAddress local_ip(192, 168, 1, 17);
IPAddress gateway(192, 168, 1, 1);
IPAddress subnet(255, 255, 255, 0);
IPAddress primaryDNS(8, 8, 8, 8);
IPAddress secondaryDNS(8, 8, 4, 4);

// MQTT
const char* mqtt_server = "192.168.1.21";
const int mqtt_port = 1883;
const char* mqtt_topic = "SENSOR_DHT22";
const char* temp_topic = "TEMP_DHT22";
const char* status_topic = "estado/DHT22";
const char* clientID = "DHT22_ESP32";
```

Usando el sensor de temperatura DHT22 se procedió con la creación de objetos para la conexión y temporización que enviara las lecturas y la señal de estado del DHT22 cada 4 y 5 segundos al bróker MQTT como se muestra en la Figura 23.

Figura 23

Creación de objetos para la conexión y temporización

```
WiFiClient espClient;
PubSubClient client(espClient);

// Temporizadores
unsigned long lastSendTime = 0;
unsigned long lastStatusTime = 0;
const int reportInterval = 4000;
const int statusInterval = 5000;
```

En la Figura 24, se configuró la función SETUP para que se ejecute una sola vez al iniciar el ESP32 el cual realizara la inicialización del puerto serial, inicializa el sensor DHT22, se conecta a la red WiFi, configura el servidor MQTT y establece conexión inicial.

Figura 24

Configuración de SETUP

```
void setup() {
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);
  dht.begin();

  connectWiFi();
  client.setServer(mqtt_server, mqtt_port);
  connectMQTT();
}
```

En la Figura 25, se configuró la función loop(), la cual se encarga de verificar si el cliente MQTT está conectado; en caso contrario, intenta reconectarse. La función client.loop() mantiene activa la comunicación con el bróker. Cada 4 segundos se realiza la lectura de humedad y temperatura desde el sensor; si la lectura es válida, se convierte a texto (payload) y se publica en el tópico correspondiente. Además, también se envía la señal de estado al tópico asignado para dicho propósito.

Figura 25

Configuración de LOOP

```
client.loop();

unsigned long now = millis();

// Enviar humedad y temperatura
if (now - lastSendTime >= reportInterval) {
  lastSendTime = now;

  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();

  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Error al leer el sensor DHT22");
    return;
  }

  char hum_payload[10];
  char temp_payload[10];
  dtostrf(humidity, 4, 1, hum_payload);
  dtostrf(temperature, 4, 1, temp_payload);

  if (client.publish(mqtt_topic, hum_payload)) {
    Serial.print("Humedad enviada: ");
    Serial.print(hum_payload);
    Serial.println(" %");
  }

  if (client.publish(temp_topic, temp_payload)) {
    Serial.print("Temperatura enviada: ");
    Serial.print(temp_payload);
    Serial.println(" °C");
  }
}

// Enviar estado "ON"
if (now - lastStatusTime >= statusInterval) {
  lastStatusTime = now;
  client.publish(status_topic, "ON", true);
}

delay(100);
}
```

Para la conexión a la red Wifi se usó la función connectWiFi(),esta función intenta conectar el ESP32 a la red WiFi usando el SSID y la contraseña definidos como se muestra en la Figura 26.

Figura 26

Configuración de connectWiFi()

```
void connectWiFi() {
  WiFi.disconnect(true);
  WiFi.mode(WIFI_STA);
  WiFi.config(local_ip, gateway, subnet, primaryDNS, secondaryDNS);
  WiFi.begin(ssid, password);
  Serial.print("Conectando a WiFi");

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("\nConectado! IP: " + WiFi.localIP().toString());
}
```

En la Figura 27, se estableció la conexión con el bróker MQTT, se utilizó la función connectMQTT(). Esta función intenta conectar el ESP32 al bróker cada 5 segundos en caso de que la conexión fallara o se hubiera perdido. De esta manera, se garantizaba una reconexión automática y continua.

Figura 27

Configuración de connectMQTT()

```
void connectMQTT() {
  while (!client.connected()) {
    Serial.print("Conectando a MQTT...");

    if (client.connect(clientID)) {
      Serial.println("Conectado!");
      client.publish(status_topic, "ON", true);
      digitalWrite(ledPin, HIGH);
    } else {
      Serial.print("Fallo, rc=");
      Serial.print(client.state());
      Serial.println(" Reintentando en 5 segundos...");
      delay(5000);
    }
  }
}
```

La configuración de los Esp32 con sus respectivos sensores restantes se puede apreciar en el Anexo 4.

3.2.4.3. Rangos de funcionamiento

En la Tabla 12, se definió los parámetros de funcionamiento del prototipo ante eventuales heladas

Tabla 13

Rango de funcionamiento de los variables

Variable	Sensor	Unidad	Rango de Activación	Justificación
Temperatura ambiente	DS18B20	°C	< 2°C	Las heladas dañan severamente el cultivo de papa si bajan de 5°C.
Temperatura del cielo	MLX90614	°C	< -10°C	El sensor mide menor temperatura cuando el cielo está despejado porque recibe menos radiación infrarroja.
Velocidad del viento	Anemómetro	km/h	< 5 km/h	Viento bajo permite mayor enfriamiento por radiación, favoreciendo heladas.
Humedad del suelo	Sensor capacitivo V1.2	%	<30%	Suelo húmedo retiene mejor el calor.
Humedad relativa	DHT22	%	< 60%	Humedad baja aumenta el riesgo de enfriamiento por pérdida de calor.

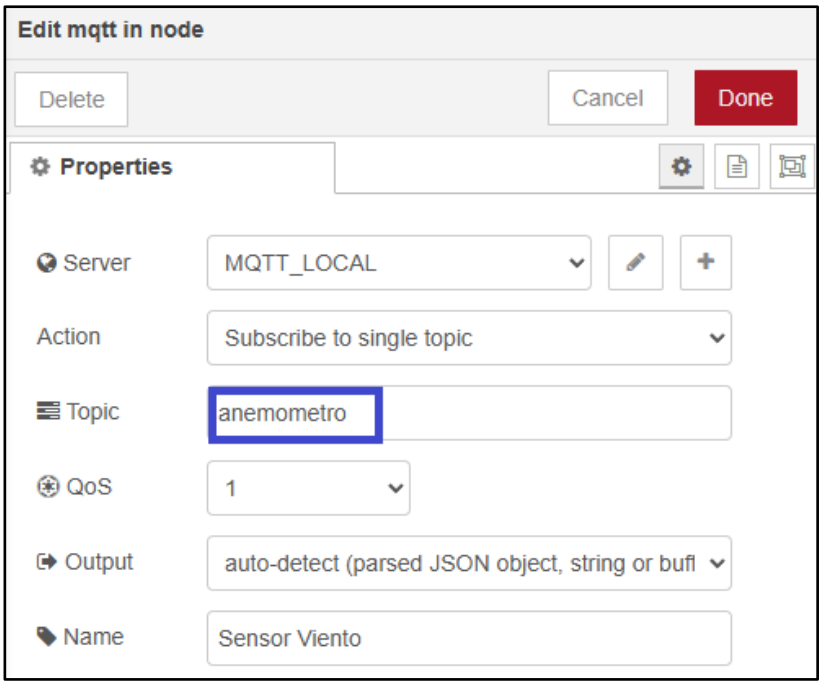
3.2.4.4. Configuración de NODE RED

- **Lectura de sensores**

  Para la lectura y visualización de los datos de los sensores en Node-RED, se utilizaron nodos *MQTT IN*, los cuales reciben la información enviada por los dispositivos ESP32 a través de los tópicos asignados a cada uno. Los tópicos utilizados corresponden a los identificadores individuales de cada ESP32. En la Figura 28, se asignó el tópico anemómetro al nodo MQTT IN para su recepción.

Figura 28

Recepción de datos en Node-RED



En la Tabla 13, se puede apreciar los tópicos e identificadores correspondientes a cada Esp32.

Tabla 14

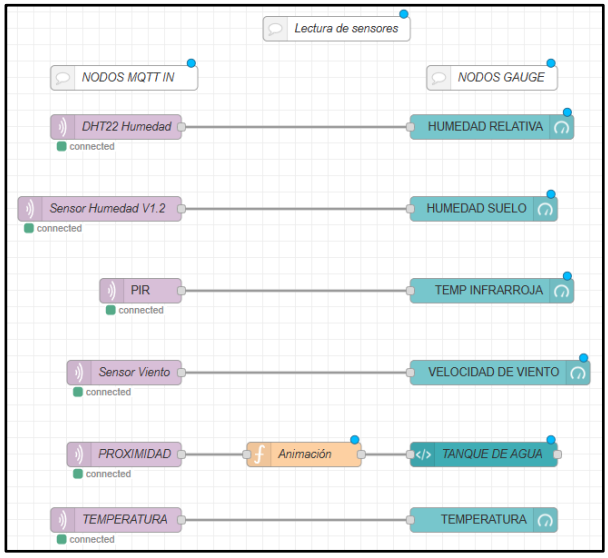
Tópicos e id de los Esp32

Tópicos	ID ESP32
Sensor_DHT22	DHT22_ESP32
Sensor_Humedad	Humedad_ESP32
PIR	PIR_ESP32
Anemómetro	Anemómetro_ES32
Proximidad	Proximidad_ESP32
Temperatura	Temperatura_ESP32

Para la visualización de las lecturas se utilizaron nodos Gauge, así como nodos *Function* y *Template*, los cuales permitieron representar gráficamente los datos y generar la animación del tanque de agua como se muestra en la Figura 29.

Figura 29

Conexión de nodos para la visualización de lecturas



- **Visualización de estados de ESP32**

Para la visualización del estado de los ESP32, se les asignó un tópico específico de estado. Cada dispositivo envía, cada 5 segundos, un mensaje con el valor “ON” a un nodo *Trigger*, el cual permite el paso del mensaje y activa un nodo *ui_LED*. Si el nodo *Trigger* no recibe un nuevo mensaje en un intervalo de 10 segundos, este envía automáticamente una señal con el valor “OFF” al nodo *ui_LED*. De esta manera, es posible detectar si alguno de los ESP32 ha dejado de transmitir, lo que podría indicar una falla en el dispositivo o en la comunicación.

En la Tabla 14, se puede apreciar los tópicos de estados correspondientes cada esp32.

Tabla 15

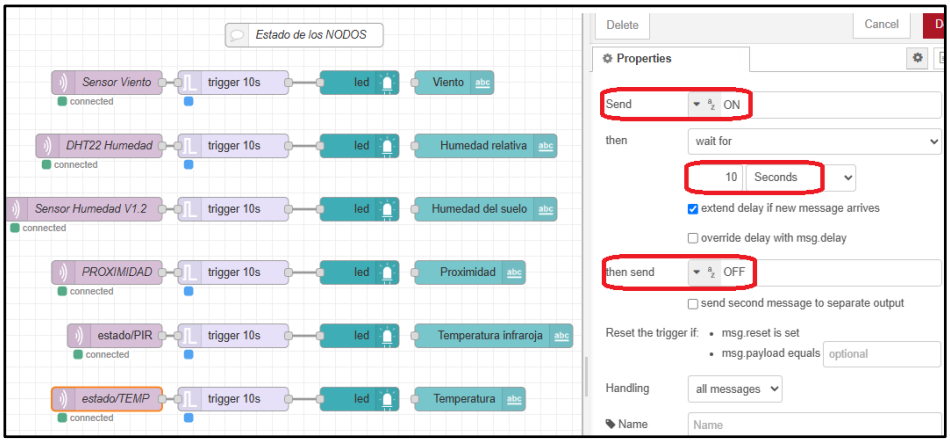
Tópicos de estado e id de los Esp32

Tópicos	ID ESP32
Estado/DHT22	DHT22_ESP32
Estado/Humedad	Humedad_ESP32
Estado/PIR	PIR_ESP32
Estado/Anemómetro	Anemómetro_ESP32
Estado/Proximidad	Proximidad_ESP32
Estado/Temperatura	Temperatura_ESP32

La Figura 30 muestra la conexión de nodos y asignación de los tópicos de estado.

Figura 30

Conexión y configuración de nodos para la visualización de estados



- **Base de datos**

Para el almacenamiento de lecturas de los sensores se utilizaron los *nodos InfluxDB Out,* los cuales se encargan de enviar y guardar los datos en la base de datos previamente creada. Asimismo, se empleó los nodos *InfluxDB In*, el cual permite consultar y recuperar los datos almacenados para su visualización.

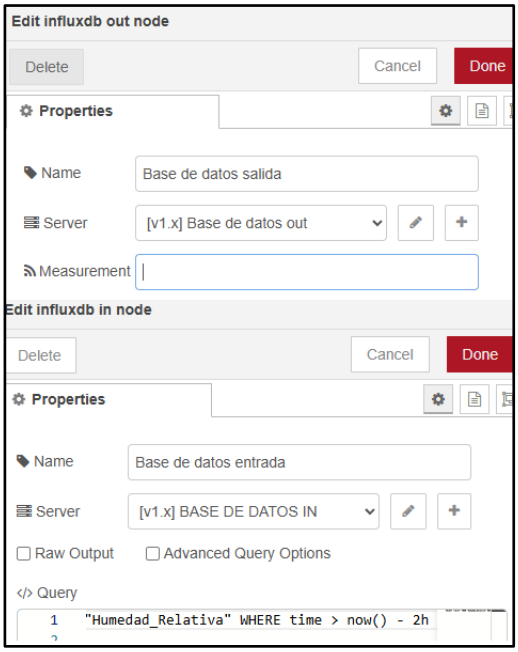
Los nodos MQTT IN (sensores) envían sus lecturas a un nodo *Function*, el cual se encarga de convertir los datos desde formato numérico a objeto y asignar el nombre

de la medición correspondiente (como Temperatura, Humedad, Viento, etc.). Una vez realizada esta conversión y asignación, la información es enviada al nodo *InfluxDB Out* para su almacenamiento en la base de datos. Posteriormente, el nodo *InfluxDB* In consulta las lecturas almacenadas y las envía a un nodo *Function*, que se encarga de darles el formato adecuado para su visualización en un nodo Chart. Para permitir una visualización continua y actualizada, se utilizó un nodo *Inject* conectado al nodo *InfluxDB In*, el cual envía una señal cada 5 segundos para que este realice una consulta de los datos correspondientes a las últimas dos horas.

En la Figura 31, se configuró el nodo influx out e In, para el almacenamiento y consulta de datos.

Figura 31

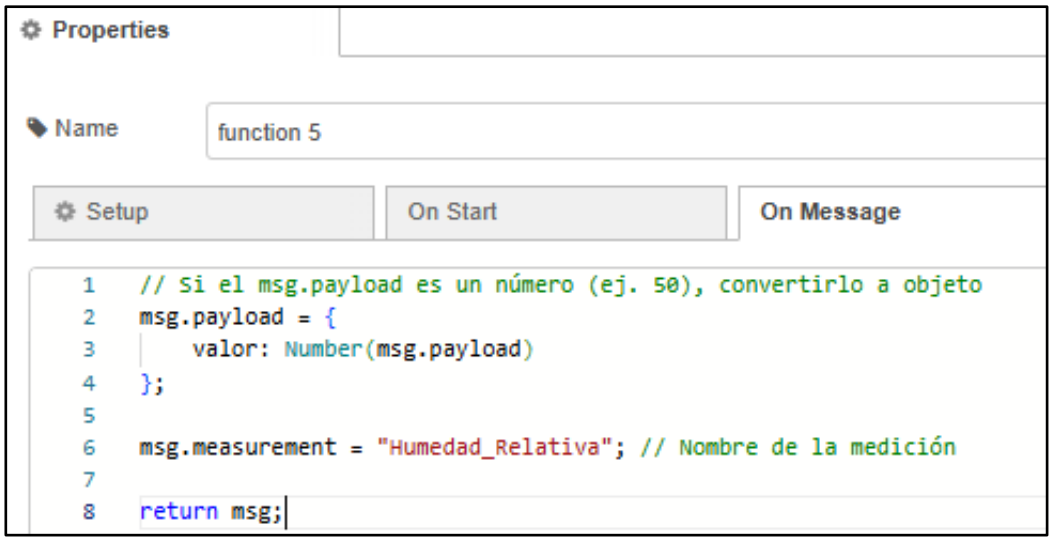
Configuración de nodos Influx Out e In



En la Figura 32, se aprecia la configuración del nodo *Function*, el cual realiza la conversión y asignación de nombres de los valores medidos.

Figura 32

Configuración de nodo Function para la conversión y asignación de mediciones



La Figura 33 y 34 muestran la conexión de nodos para el almacenamiento y la visualización de los datos correspondientes a los sensores DHT22, de humedad de suelo, temperatura infrarroja, temperatura y viento.

Figura 33

Conexión de nodos para sensores DHT22, humedad, temperatura infrarroja

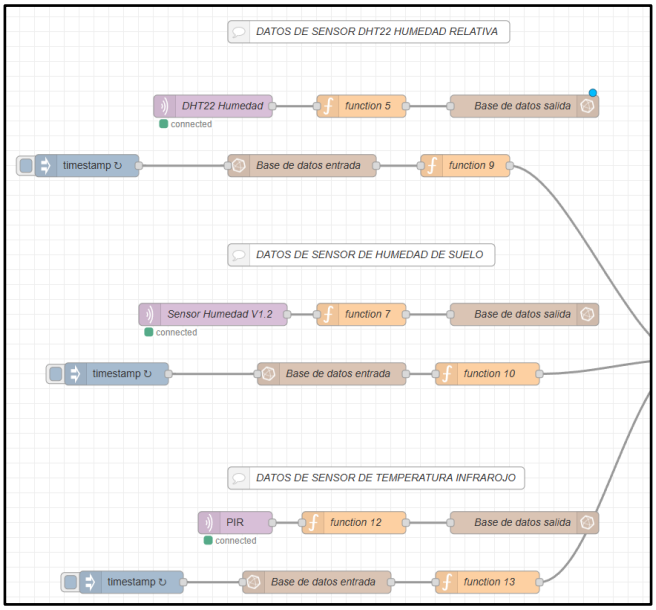
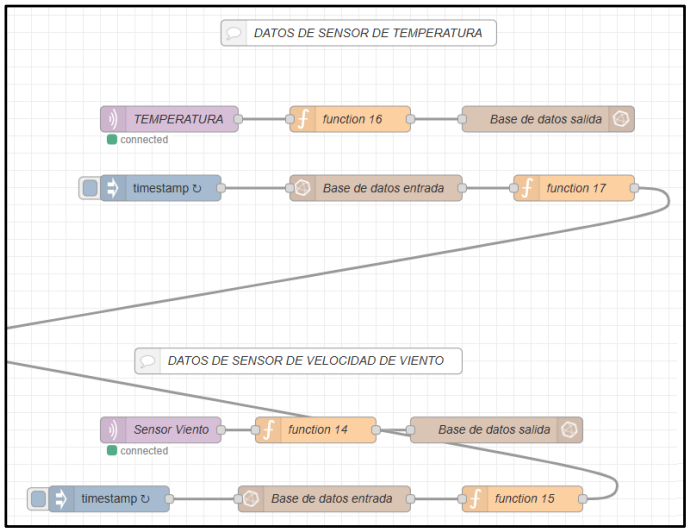


Figura 34

Conexión de nodos para sensores viento y temperatura



Los datos se integran mediante un nodo *Join*, el cual combina las lecturas provenientes de los distintos sensores. Estas lecturas unificadas se envían a un nodo *Function*, que las ordena y estructura en el formato adecuado para su posterior visualización en un nodo *Chart*, como se muestra en las Figuras 35 y 36.

Figura 35

Configuración de nodo Function para la visualización de las lecturas

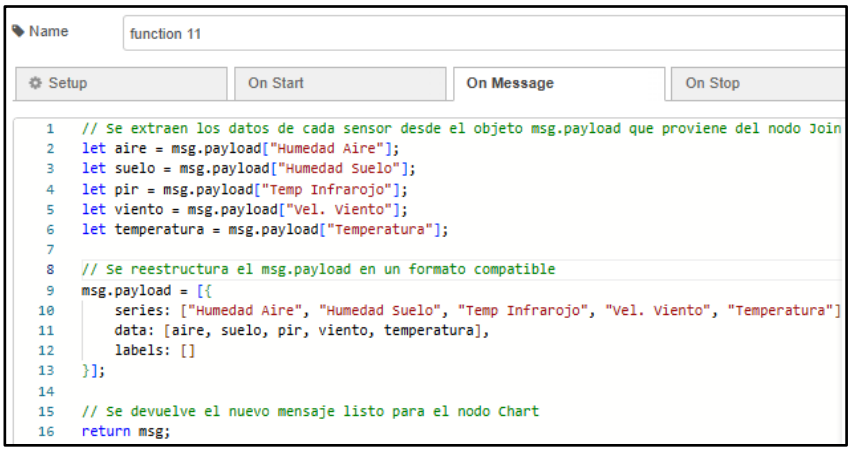
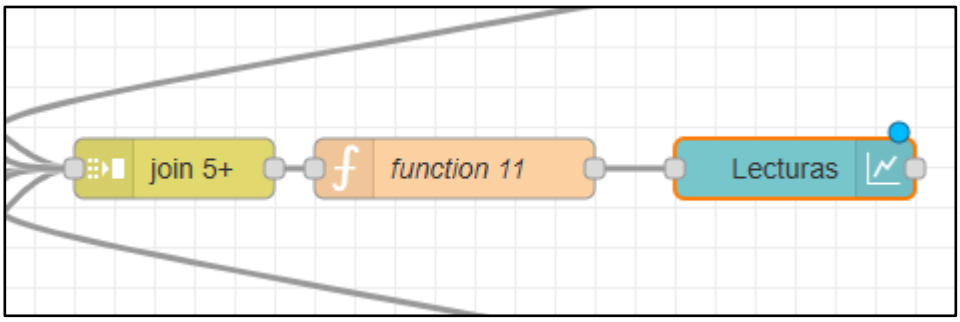


Figura 36

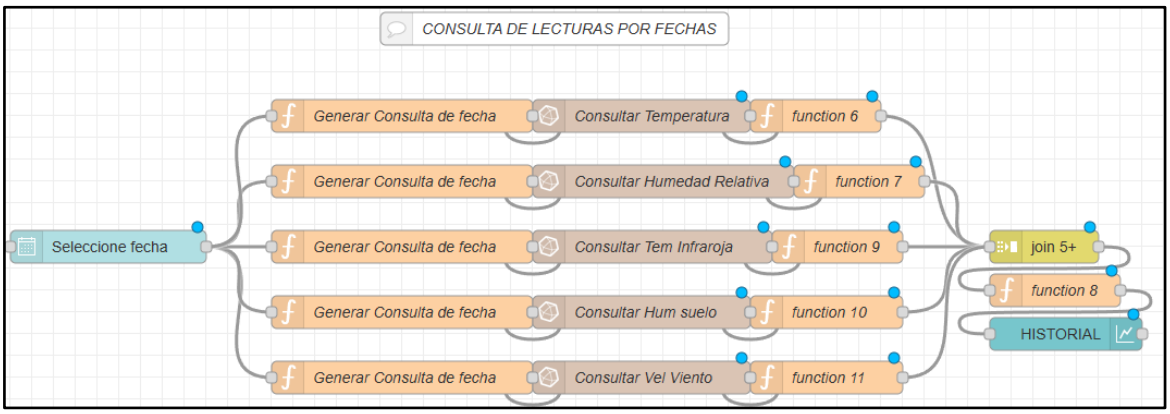
Conexión de nodos Join, Function y chart para la visualización de las lecturas



Para la consulta de lecturas correspondientes a fechas anteriores, se utilizó un nodo Date *Picker*, el cual permite seleccionar una fecha específica. Además, se empleó un nodo *Function* encargado de ajustar el formato de la fecha seleccionada y generar la consulta adecuada para enviarla al nodo *InfluxDB* In. Finalmente, se utilizaron los nodos *Join* y *Function* previamente mencionados, encargados de organizar y preparar los datos para su visualización en el nodo *Chart*, como se muestra en la Figura 37.

Figura 3

Conexión de nodos Date Picker, Influx In para la visualización de lecturas



- **Condiciones de activación ante helada inminente**

Para la activación del sistema de riego por aspersión contra heladas, deben cumplirse las siguientes condiciones: la velocidad del viento debe ser menor a 5 km/h, la temperatura debe ser igual o inferior a 2 °C y el nivel de agua no debe estar por debajo del mínimo establecido. En caso de que el sensor de temperatura principal falle, se considerarán las lecturas de un sensor de respaldo hasta que el sensor principal vuelva a funcionar. El sistema de riego permanecerá activo hasta 15 minutos después de que la temperatura supere nuevamente los 5 °C.

En la figura 38, se configuró un nodo Function el cual se encarga de recibir y registrar las lecturas de los nodos MQTT In.

Figura 38

Registro de lecturas de nodos MQTT In

```
let ahora = Date.now();

// === Guardar valores y timestamps ===
if (msg.topic === "TEMPERATURA") {
    context.set("temperatura", parseFloat(msg.payload));
    context.set("timestamp_temperatura", ahora);
    return null;
} else if (msg.topic === "TEMP RESPALDO") {
    context.set("pir", parseFloat(msg.payload));
    context.set("timestamp_temrespaldo", ahora);
    return null;
} else if (msg.topic === "ANEMOMETRO") {
    context.set("viento", parseFloat(msg.payload));
    context.set("timestamp_viento", ahora);
    return null;
} else if (msg.topic === "PROXIMIDAD") {
    context.set("distancia", parseFloat(msg.payload));
    context.set("timestamp_distancia", ahora);
} else {
    return null; // Ignora otros tópicos
}
```

En la Figura 39, se recuperaron los valores y marcas de tiempo previamente guardados para cada sensor, con el fin de verificar si aún son recientes.

Figura 39

Recuperación de valores de lecturas de los sensores

```
// === Obtener valores ===
let temp = context.get("temperatura");
let tiempo_temp = context.get("timestamp_temperatura") || 0;

let pir = context.get("temrespaldo");
let tiempo_pir = context.get("timestamp_temrespaldo") || 0;

let viento = context.get("viento");
let tiempo_viento = context.get("timestamp_viento") || 0;

let distancia = context.get("distancia");
let tiempo_distancia = context.get("timestamp_distancia") || 0;

let estadoAnterior = context.get("estadoAnterior") || "OFF";
let tiempoActivacion = context.get("tiempoActivacion");
```

Si la lectura del sensor de temperatura principal es válida, se utiliza. En caso contrario, se toma la lectura del sensor de respaldo, como se puede apreciar en la Figura 40.

Figura 40

Selección de sensor de temperatura

```
// === Selección de sensor de temperatura ===
let tempUsada = undefined;
if (tempValida) {
    tempUsada = temp; // Sensor principal
} else if (temrespaldoValida) {
    tempUsada = temrespaldo;  // Sensor de respaldo
} else {
    return null; // No hay temperatura válida
}
```

La figura 41 muestra que, si la distancia medida por el sensor ultrasónico es alta indicando tanque bajo o vacío, se apaga el sistema si estaba encendido, ya que no hay agua suficiente para el riego.

Figura 4

Verificación del nivel de agua

```
55  // === Verificación del nivel de agua ===
56  // Si la distancia es alta, (detener sistema)
57  if (distanciaValida && distancia >= 90) {
58      if (estadoAnterior === "ON") {
59          context.set("estadoAnterior", "OFF");
60          msg.payload = "OFF";
61          return msg;
62      } else {
63          return null; // Ya está apagado, no hacer nada
64      }
65  }
```

En la Figura 42, se estableció las condiciones bajo las cuales el sistema puede activarse, si las condiciones dejan de cumplirse, se espera 10 segundos antes de apagar el sistema, permitiendo una desactivación controlada.

Figura 42

Evaluación de condiciones y desactivación del sistema

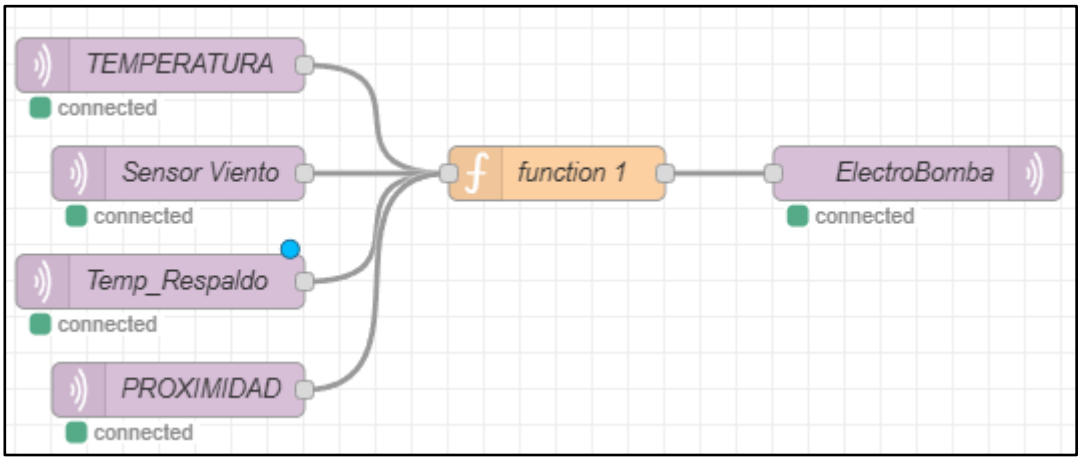
```
// === Evaluar condiciones de activación ===
if (!vientoValido) return null; // Viento no válido

let condiciones = (tempUsada <= 30 && viento < 2);

// === Control de estado ===
if (condiciones) {
    context.set("tiempoActivacion", ahora);
    if (estadoAnterior !== "ON") {
        context.set("estadoAnterior", "ON");
        msg.payload = "ON";
        return msg;
    }
} else {
    if (estadoAnterior === "ON" && tiempoActivacion !== undefined) {
        let tiempoTranscurrido = ahora - tiempoActivacion;
        if (tiempoTranscurrido >= 10000) {
            context.set("estadoAnterior", "OFF");
            msg.payload = "OFF";
            return msg;
        }
    }
}

return null;
```

Una vez establecido el funcionamiento en el nodo *Function*, se conecta un nodo *MQTT Out* encargado de transmitir la señal de activación mediante el tópico “VALVULA”. Esta señal es enviada al microcontrolador ESP32, el cual se encuentra vinculado a los actuadores del sistema, permitiendo así la activación del sistema de riego por aspersión. Adicionalmente, para monitorear el estado de la válvula, se utiliza el tópico “estado/VALVULA”, a través del cual el ESP32 publica el estado actual del actuador, facilitando así la supervisión en tiempo real desde la plataforma Node-RED.

La Figura 43 muestra la conexión de nodos MQTT In, Function y MQTT Out, para la activación de la electroválvula.

Figura 5

Conexión de nodos MQTT In, Function y MQTT Out



Ante la detección de una posible helada, el sistema de riego por aspersión se activa de forma preventiva y se mantiene encendido hasta que la humedad del suelo supere el 35 %, ya que un suelo húmedo ayuda a conservar y liberar calor. Para su activación deben cumplirse las siguientes condiciones:

- Velocidad del viento menor a 5 km/h
- Humedad relativa menor al 60 %
- Temperatura infrarroja del cielo menor a -10 °C
- Humedad del suelo menor al 35 %

Los pasos para la implementación de estas condiciones en la programación del nodo Function en Node-RED se detallan en el Anexo 5.

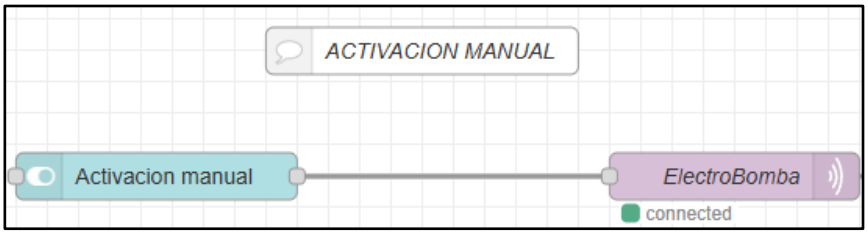
- **Activación manual**

Para permitir la activación manual del sistema de riego, se implementó un nodo switch que envía una señal "ON" al nodo mqtt out, el cual publica en el tópico "BombaAgua" correspondiente al actuador. Esta acción permite forzar el encendido del riego sin considerar las condiciones ambientales. De igual forma, es posible enviar una señal "OFF" para desactivarlo manualmente.

La figura 44 muestra la conexión de los nodos SWITCH y MQTT OUT para la activación manual de la Bomba de agua.

Figura 44

Activación manual



- **Envío de alertas a Telegram**

Con el objetivo de mejorar el monitoreo y la capacidad de respuesta del sistema, se implementó un mecanismo de notificaciones automáticas mediante la integración con la plataforma de mensajería Telegram. Estas notificaciones son generadas cuando se detecta un nivel de agua por debajo del umbral mínimo establecido, cuando se cumplen

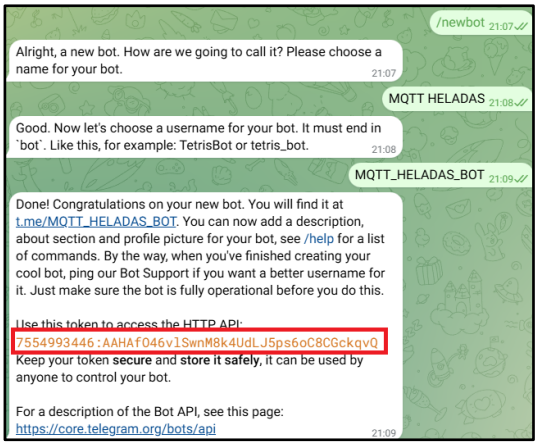
las condiciones ambientales que indican la posible ocurrencia de una helada o helada inminente.

Para la implementación del sistema de notificaciones, se creó un bot en Telegram utilizando el bot oficial de configuración llamado 'BotFather'. El proceso inició con el comando /newbot, tras lo cual se solicitó asignar un nombre al bot, eligiéndose “MQTT HELADAS”. Posteriormente, se definió un nombre de usuario único que terminara en 'bot', registrándose como “MQTT_HELADAS_BOT”. Al finalizar este proceso, Telegram generó un *token* de autenticación, necesario para integrar y utilizar el bot en plataformas externas como Node-RED, permitiendo así el envío automatizado de mensajes al usuario ante eventos relevantes del sistema.

En la Figura 45, se puede apreciar los pasos que se realizó para la creación del bot de Telegram.

Figura 6

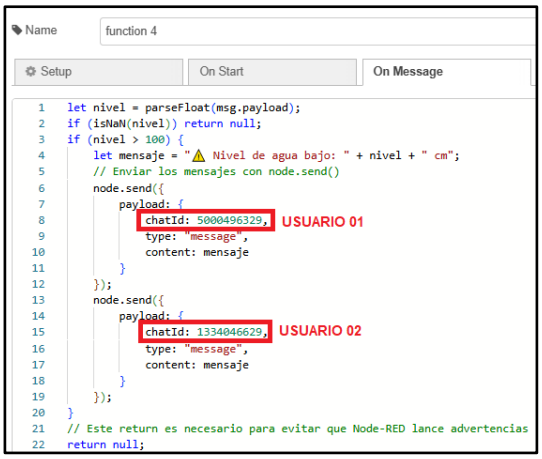
Creación del bot de Telegram



Seguidamente, se configuró un nodo *Function* que verifica el nivel de agua recibido y, en caso de que este supere un límite establecido, genera una alerta automática que es enviada por Telegram a dos usuarios específicos, utilizando sus respectivos IDs, como se puede apreciar en la Figura 46.

Figura 7

Verificación del nivel de agua y envío de alerta a usuarios



Por último, se procede a configurar un nodo Telegram Sender, el cual se encarga de enviar las notificaciones. Para ello, se le asigna el nombre del bot y el token generado previamente durante la creación del bot en Telegram.

En la figura 47 y 48, se puede apreciar la configuración de los nodos Telegram Sender y la conexión de los nodos para el envío de notificaciones a Telegram.

Figura 47

Configuración del nodo Telegram Sender

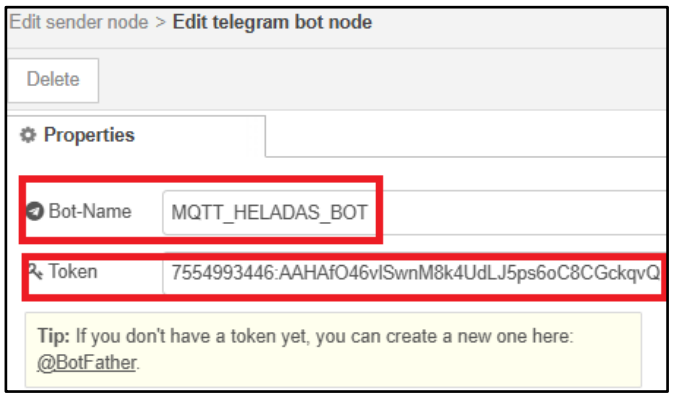
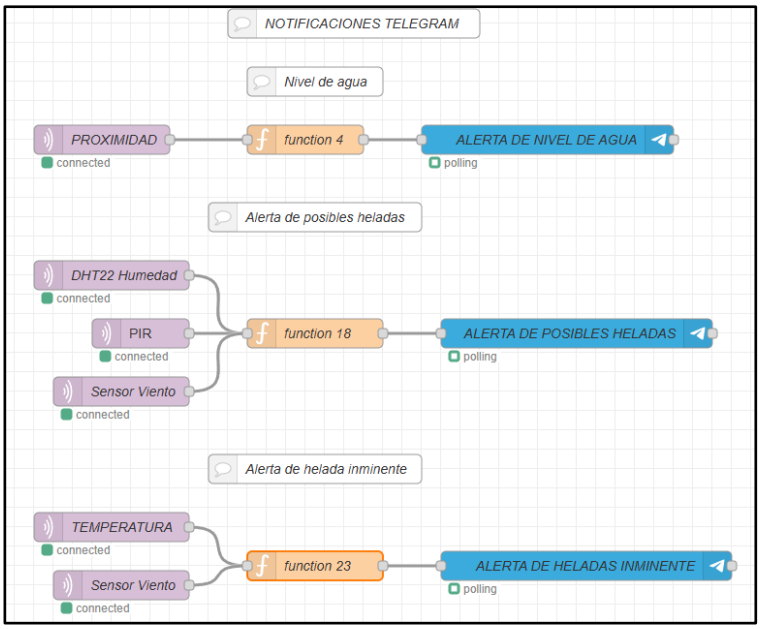


Figura 48

Conexión de nodos para el envío de notificaciones a Telegram



También se configuraron notificaciones para heladas inminentes y posibles heladas. La implementación de estas notificaciones se detalla en el Anexo 9.

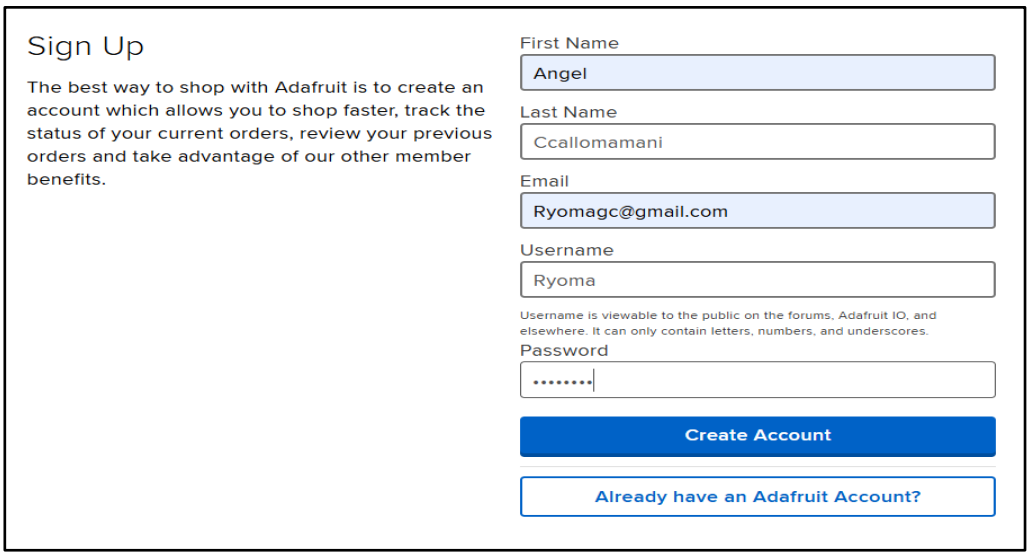
- **Monitorización mediante ADAFRUIT IO**

Para el monitoreo remoto se optó por la plataforma Adafruit IO, debido a que es altamente compatible con el protocolo MQTT utilizado por el broker Mosquitto y permite una integración sencilla con Node-RED. Esta plataforma proporciona dashboards personalizables para la visualización de datos en tiempo real, lo cual facilita el seguimiento del sistema desde cualquier ubicación con acceso a internet. Además, su interfaz intuitiva y soporte para dispositivos IoT como el ESP32 la convierten en una herramienta eficiente y adecuada para proyectos de monitoreo y control automatizado."

La Figura 49 muestra los pasos realizados para la creación de una cuenta en Adafruit.

Figura 8

Creación de cuenta en Adafruit IO



Se crearon *Feeds* para almacenar los datos; los cuales actúan como contenedores individuales para cada tipo de información que se desea registrar o visualizar. Para ello, se selecciona la opción *Feeds* en el menú principal de Adafruit IO y luego *Create a New Feed*. A cada feed se le asigna un nombre, estos nombres serán utilizados posteriormente para establecer la conexión MQTT desde Node-RED.

En la Tabla 15, se puede visualizar los *FEED* Y *KEYS MQTT* necesarios para la comunicación entre Adafruit IO y Node Red

Tabla 16

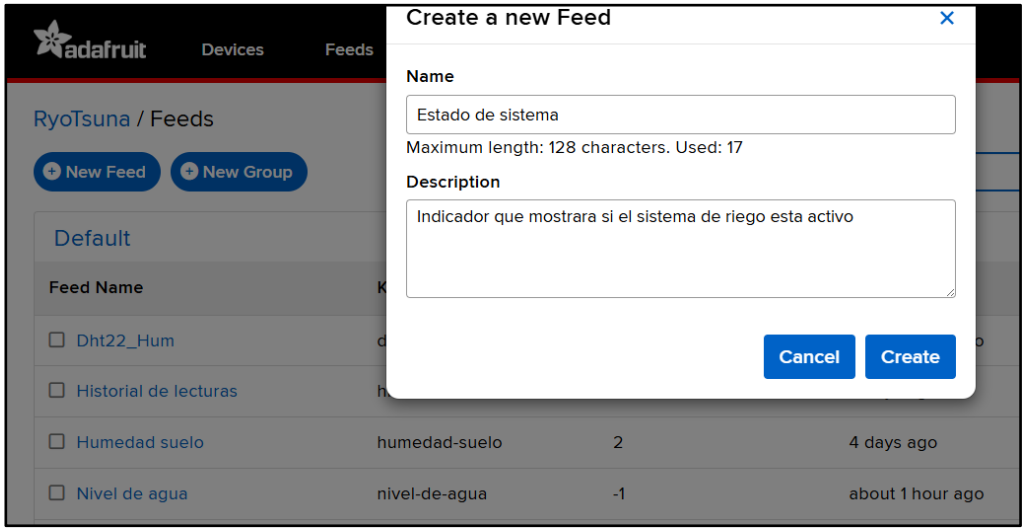
Feeds y keys

Feed	Key MQTT
DHT22_Hum	RyoTsuna/feeds/DHT22-hum
Historial de lecturas	RyoTsuna/feeds/Historial-de-lecturas
Humedad Suelo	RyoTsuna/feeds/Humedad-suelo
Nivel de agua	RyoTsuna/feeds/Nivel-de-agua
PIR	RyoTsuna/feeds/PIR
Temperatura	RyoTsuna/feeds/Temperatura
Viento	RyoTsuna/feeds/Viento

La Figura 50 muestra la creación de los Feeds.

Figura 9

Creación de Feeds

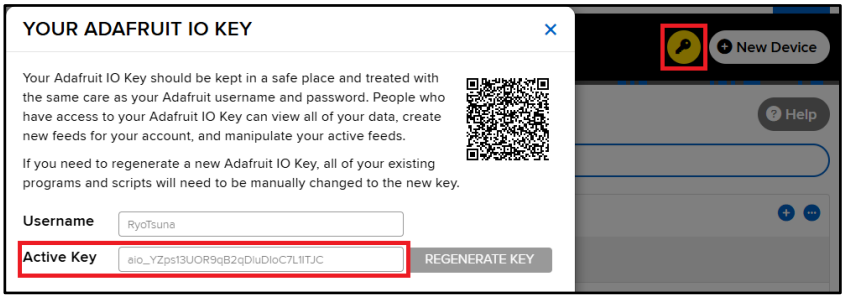


Para establecer la conexión entre Adafruit IO y Node-RED, se utilizó una clave de autenticación única denominada AIO Key. Esta clave se obtuvo al hacer clic en el nombre de usuario y seleccionar la opción "My Key", donde se mostraron tanto el nombre de usuario como la clave secreta. La AIO Key fue utilizada como contraseña al configurar el cliente MQTT, lo que permitió una comunicación segura con los feeds creados en la plataforma.

En la Figura 51, se puede apreciar la clave de autentificación única

Figura 51

Obtención de clave de autenticación AIO Key

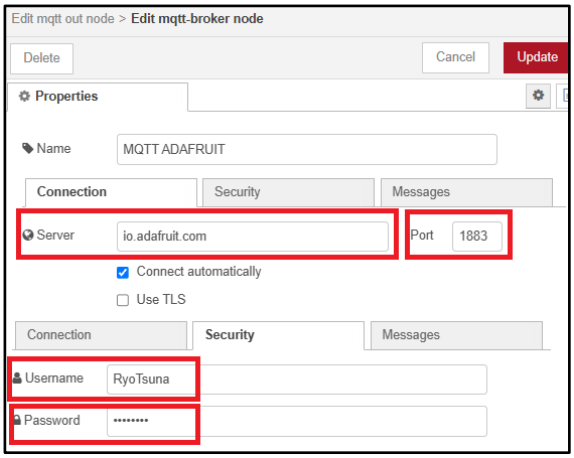


En la plataforma Node-RED, se estableció la comunicación con Adafruit IO mediante los nodos MQTT. Para ello, se configuraron un nodo MQTT out (para el envío de datos) y un nodo MQTT in (para la recepción de datos). Al hacer doble clic sobre cada nodo, se creó una nueva conexión MQTT en la que se definieron los siguientes parámetros: como servidor se ingresó io.adafruit.com, se utilizó el puerto 1883, el nombre de usuario correspondió al de la cuenta registrada en Adafruit, y en el campo de contraseña se introdujo la AIO Key obtenida previamente.

La Figura 52 muestra la configuración del nodo MQTT OUT para su conexión con Adafruit IO mediante el usuario y contraseña previamente creada.

Figura 10

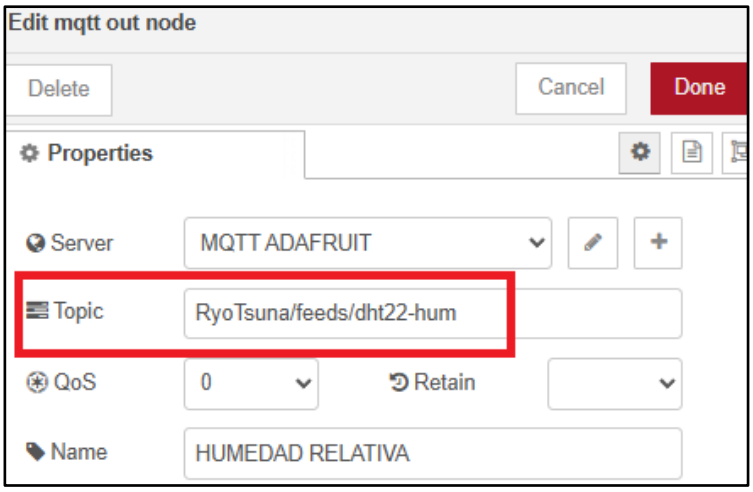
Configuración de nodos MQTT para la conexión con Adafruit IO



En la Figura 53, se ingresaron las keys de los feeds creados anteriormente. Una vez completada esta configuración, los nodos se conectaron automáticamente con la nube de Adafruit IO, permitiendo así la integración con el sistema.

Figura 11

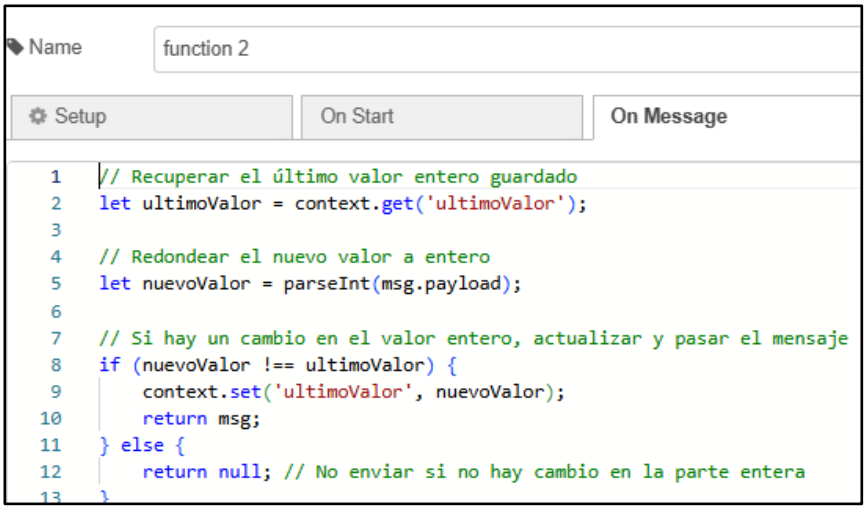
Configuración de tópicos en nodos MQTT



Debido a las limitaciones de la versión gratuita de la plataforma Adafruit IO, la cual permite la recepción de hasta 30 mensajes por minuto, se configuró un nodo Function en Node-RED que solo permite el envío de lecturas cuando se detecta un cambio en los valores. De esta manera, se evita la saturación del sistema y se optimiza el uso del canal de comunicación, como se muestra en la Figura 54.

Figura 12

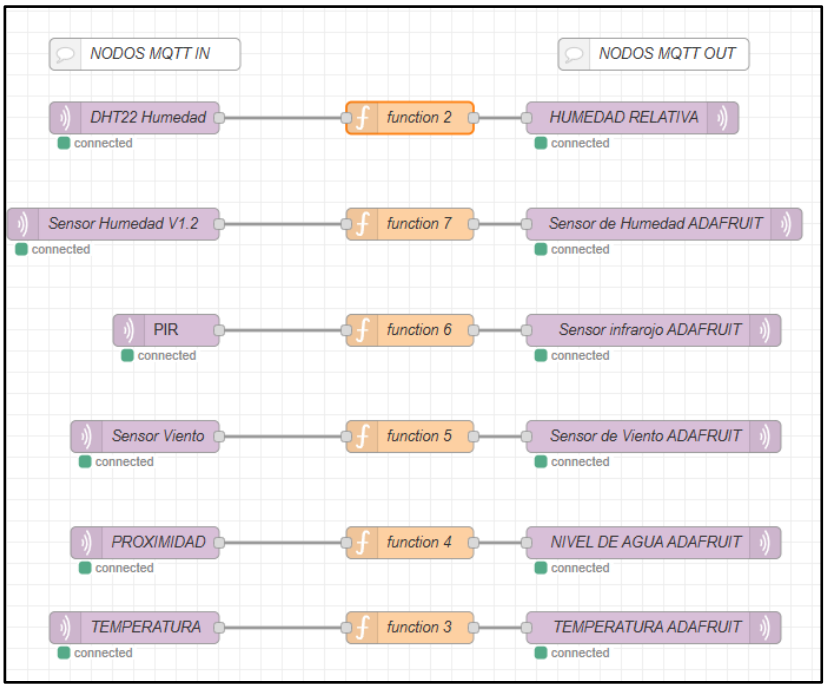
Configuración de nodo Function para el envío de lecturas a Adafruit



En la Figura 55, se puede apreciar la conexión de los nodos para la comunicación entre Node Red y Adafruit IO.

Figura 13

Conexión de nodos para el envío y recepción de datos a Adafruit



3.3. Materiales e instrumentos

3.3.1. Microcontrolador ESP32

El ESP32 es un sistema en chip (SoC) desarrollado por Espressif Systems, ideal para proyectos de IoT, ya que combina Wi-Fi y Bluetooth BLE. Con Wi-Fi, permite conectarse a redes e internet, mientras que el Bluetooth facilita la conexión directa con dispositivos como celulares. Es eficiente en energía, con un consumo muy bajo en reposo, lo que lo hace perfecto para dispositivos portátiles con batería. Incluye dos núcleos de procesador ajustables y soporta una amplia gama de sensores y conexiones. Funciona a 3.3 V, por lo que no debe usarse con 5 V, y se recomienda agregar un capacitor para estabilizar la alimentación (Electronilab, s.f.)

En la Figura 56, podemos observar la distribución de pines de un Esp32 de 38 pines.

Figura 14

Distribución de pines del microcontrolador ESP32

Nota. Adaptado de *Módulo ESP32 con WiFi y Bluetooth ESP32 38 pines ESP-WROOM-32 alta capacidad*, del sitio web Electrónica Paraguay (s.f.), en https://www.electronica.com.py/producto/modulo-esp32-con-wifi-y-bluetooth-esp32-38-pines-esp-wroom-32-alta-capacidad/

La Tabla 16 muestra las especificaciones técnicas del microcontrolador ESP32.

Tabla 17

Especificaciones del microcontrolador ESP32

Características	Detalles
Procesador	Xtensa® dual-core 32-bit LX6
Frecuencia de reloj	Hasta 240 MHz
Memoria SRAM	500 KB
Almacenamiento	Soporte para memoria flash externa
Conectividad inalámbrica	Wi-Fi 802.11 b/g/n y Bluetooth 4.2 BR/EDR y BLE
Periféricos	Múltiples interfaces incluyendo SPI, I2C, UART, ADC, DAC, PWM, entre otros
Seguridad	Soporte para cifrado AES, RSA, SHA, y gestión de claves
Consumo de energía	80 mA promedio a 200 mA max
Rango de temperatura	-40°C a +125°C

3.3.2. Sensor de temperatura DS18B20

El DS18B20 es un sensor de temperatura digital que utiliza el protocolo de comunicación 1-Wire. Este sensor puede medir temperaturas en el rango de -55°C a +125°C con una

precisión de ±0.5°C. Cada sensor tiene un código serial único de 64 bits, lo que permite la conexión de múltiples sensores en una misma línea de datos. Este sensor es ampliamente utilizado en aplicaciones de monitoreo ambiental y sistemas de climatización (Analog Devices, s.f.).

La Tabla 17 muestra las especificaciones técnicas del sensor de temperatura DS18B20.

Tabla 18

Especificaciones del sensor de temperatura DS18B20

Características	Detalle
Rango de operación	-55°C a +125°C
Precisión	±0.5°C en -10°C a +85°C
Resolución	Configurable de 9 a 12 bits
Voltaje de operación	3V a 5V
Corriente de operación	1mA
Tiempo de conversión	93.75 ms (9 bits) a 750 ms (12 bits)
Formato de salida	Digital
Modo de alimentación	Normal
Interfaz de comunicación	1-Wire

En la Figura 57, podemos observar un sensor de temperatura DS18B20 con su blindaje de protección.

Figura 15
Sensor de temperatura DS18B20

Nota. Adaptado de *Sensor de Temperatura Digital DS18B20*, del sitio web Naylamp Mechatronics (s.f.), en https://naylampmechatronics.com/sensores-temperatura-y-humedad/16-sensor-de-temperatura-digital-ds18b20.html

3.3.3. Sensor de temperatura y humedad DHT22

El DHT22, es un sensor digital de temperatura y humedad relativa que opera en un rango de -40°C a 80°C con una precisión de ±0.5°C y mide humedad entre 0% y 100% HR con una precisión de ±2%. Utiliza un protocolo de comunicación digital de un solo cable, requiere un voltaje de operación de 3.3V a 5.5V y consume 1.5 mA durante la medición (Electropeak, 2010).

La Tabla 18 muestra las especificaciones técnicas del sensor DHT22

Tabla 19

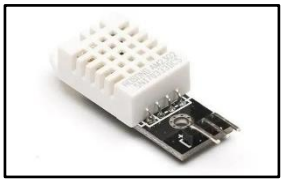
Especificaciones del sensor de temperatura y humedad DHT22

Características	Detalles
Alimentación	3.3V
Rango de medición de temperatura	-40°C a 80°C
Precisión	±0.5 °C
Resolución temperatura	0.1°C
Rango de medición de humedad	0 a 100% HR
Precisión de medición de humedad	2% HR
Resolución de humedad	0.1% HR
Tiempo de lectura	2 Seg

En la Figura 58, podemos observar un sensor DHT22 utilizado para medir temperatura y humedad relativa.

Figura 16

Sensor de Humedad y Temperatura DHT22



Nota. Adaptado de *Sensor de Humedad y Temperatura DHT22*, del sitio web MCI Electronics (s.f.), en https://mcielectronics.cl/shop/product/sensor-de-humedad-y-temperatura-dht22/

3.3.4. Sensor ultrasónico HC-SR04

El sensor ultrasónico HC-SR04 mide la distancia mediante el envío de pulsos ultrasónicos y la detección de su eco. Tiene un rango de medición de 2 cm a 400 cm con una precisión de ±3 mm (Random Nerd Tutorials, s.f)

La Tabla 19 muestra las especificaciones técnicas del sensor ultrasónico HC-SR04.

Tabla 20

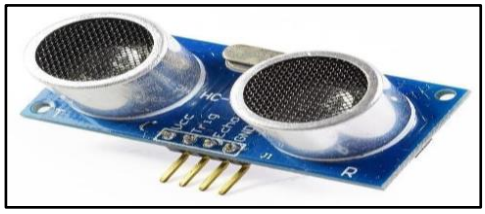
Especificaciones del sensor ultrasónico HC-SR04

Características	Detalles
Voltaje de operación	5V
Corriente de operación	15mA
Frecuencia de operación	40 Hz
Rango máximo	4m
Rango mínimo	2cm
Ángulo de medición	15°
Señal de activación	Pulso TTL de 10 µs
Señal de salida	Señal TTL proporcional a la distancia

En la Figura 59, podemos observar un Sensor ultrasónico HC-SR04 utilizado para medir el nivel de agua.

Figura 17

Sensor ultrasónico HC-SR04



Nota. Adaptado de *Sensor ultrasónico HC-SR04*, del sitio web Naylamp Mechatronics (s.f.), en https://naylampmechatronics.com/sensores-proximidad/10-sensor-ultrasonido-hc-sr04.html

3.3.5. Sensor Capacitivo de Humedad V1.2

El sensor capacitivo de humedad de suelo V1.2 es un instrumento utilizado para la medición de la humedad del suelo mediante el principio de detección capacitiva, diferenciándose de los sensores resistivos al evitar el contacto directo entre los electrodos y el medio. Su funcionamiento se basa en la variación de la capacitancia generada por la presencia de agua en el suelo, lo que permite obtener mediciones más estables y reducir los problemas asociados a la corrosión. La señal resultante es de tipo analógico y puede ser procesada por microcontroladores como el ESP32 para su interpretación y análisis. Gracias a este mecanismo, se logra una medición más precisa y una mayor vida útil del dispositivo (Naylamp Mechatronics, s.f.).

La Tabla 20 muestra las especificaciones técnicas del sensor Capacitivo de Humedad V1.2.

Tabla 21

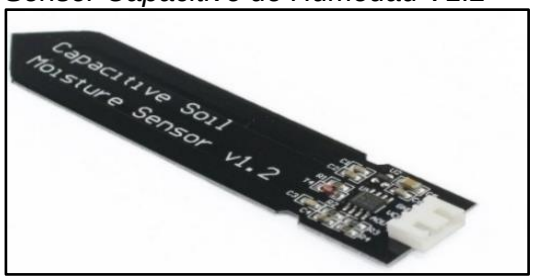
Especificaciones del sensor Capacitivo de Humedad V1.2

Características	Detalles
Alimentación	3.3V-5V
Corriente de operación	5mA
Voltaje de la señal de salida	0 a 5V analógico
Vida útil	3 años min
Conector	PH2.0-3P
Dimensiones	98*23 mm
Peso	15 gramos
Modelo	Capacitive soil moisture sensor v1.2

En la Figura 60, podemos observar un Sensor Capacitivo de Humedad V1.2.

Figura 18

Sensor Capacitivo de Humedad V1.2



Nota. Adaptado de *Sensor capacitivo de humedad V1.2*, del sitio web Naylamp Mechatronics (s.f.), en https://naylampmechatronics.com/sensores-temperatura-y-humedad/538-sensor-de-humedad-de-suelo-capacitivo-v1.html

3.3.6. Sensor de temperatura infrarrojo mlx90614

El MLX90614 es un sensor infrarrojo fabricado por Melexis que permite medir la temperatura de un objeto a distancia, sin necesidad de contacto físico. Utiliza tecnología de detección de radiación infrarroja para captar la energía térmica emitida por los objetos, procesándola internamente para entregar mediciones precisas y confiables. Es ampliamente utilizado en aplicaciones que requieren monitoreo de temperatura no invasivo, gracias a su diseño compacto y su capacidad para operar en un amplio rango de condiciones. Además, cuenta con interfaces de comunicación estándar que facilitan su integración en sistemas electrónicos y embebidos (Naylamp Mechatronics, s.f.).

La Tabla 21 muestra las especificaciones técnicas del sensor de temperatura infrarrojo MLX90614.

Tabla 22

Especificaciones del sensor de temperatura infrarrojo MLX90614

Características	Detalle
Voltaje de operación	3.3v – 5V
Protocolo de comunicación	SMBus
Rango de temperatura Ambiente	-40°C hasta +170°C
Rango de temperatura del objeto	-70°C hasta +380°C
Precisión	±0.5°C
ADC interno	17 bits
Procesador digital de señal interno	Si
Dimensiones	16 x 11 x 6 mm
Peso	2.8 gramos

En la Figura 61, podemos observar un Sensor de temperatura infrarrojo MLX90614 con sus respectivos pines de conexión.

Figura 61

Sensor de temperatura infrarrojo MLX90614

Nota. Adaptado de *Sensor de temperatura infrarrojo*, del sitio web Naylamp Mechatronics (s.f.), en https://naylampmechatronics.com/sensores-temperatura-y-humedad/330-sensor-de-temperatura-mlx90614.html

3.3.7. Raspberry pi 3b

La Raspberry Pi 3 es una computadora de bajo costo y tamaño reducido, diseñada para facilitar el acceso a la informática. Cuenta con un procesador ARM de 64 bits con cuatro núcleos a 1,2 GHz y 1 GB de memoria RAM, lo que permite ejecutar diversas tareas y el uso de múltiples sistemas operativos basados en Linux y Windows 10. Su diseño y funcionalidad la hacen adecuada para entornos educativos, el desarrollo de prototipos y aplicaciones en electrónica y programación (Profesional Review, 2021).

La Tabla 22 muestra las especificaciones técnicas de un Raspberry Pi 3.

Tabla 23

Especificaciones de la Raspberry Pi 3

Características	Detalles
Procesador	CPU ARM Cortex-A53 de 64 bits, cuatro núcleos a 1.2 GHz
Memoria RAM	1 GB LPDDR2
Conectividad	Wi-Fi 802.11n y Bluetooth 4.1 integrados
Puertos USB	4 puertos USB 2.0
Almacenamiento	Ranura para tarjeta microSD
Salidas de video	Puerto HDMI
Audio	Salida de audio estéreo de 3.5 mm
Gpio	40 pines GPIO
Alimentación	5V/2.5 mA
Dimensiones	85 x 56 x 17 milímetros

En la Figura 62, se muestra una Raspberry Pi 3 con sus puertos de entrada y salida.

Figura 62

Raspberry PI 3b

Nota. Adaptado de *Raspberry Pi 3 Model B* [Fotografía], por Raspberry Pi (s.f.), en https://www.raspberrypi.com/products/raspberry-pi-3-model-b/

3.3.8. Anemómetro de cazoletas PA2

El anemómetro de cazoletas PA2 es un sensor diseñado para medir la velocidad del viento mediante la rotación de sus cazoletas, generando una señal de salida proporcional a la velocidad del aire. Su funcionamiento se basa en el efecto Hall, lo que le permite ofrecer mediciones precisas en diversas aplicaciones meteorológicas e industriales (Sensovant, s.f.).

La Tabla 23 muestra las especificaciones técnicas de anemómetro de cazoletas PA2.

Tabla 24

Especificaciones del anemómetro de cazoletas PA2

Características	Detalles
Salida de señal	2 pulsos por rotación
Rango de medición	0 a 60 m/s
Umbral de inicio	0.5 m/s
Frecuencia de salida	67 Hz a 30 m/s
Temperatura de operación	-30 °C a +70 °C
Conexión	V+, Señal y Gnd

En la Figura 63, podemos observar un anemómetro de cazoletas.

Figura 63

Anemómetro con Sensor de velocidad del viento

Nota. Adaptado de *Anemómetro de cazoletas* [Fotografía], por Sensovant (s.f.), en https://sensovant.com/productos/meteorologia/sensores-viento-mecanicos/anemometro-de-cazoletas/

3.3.9. Bomba de agua DP-521

El modelo DP-521 es una bomba de agua de diafragma no sumergible, de 12 V DC y 2 A, diseñada para proporcionar un caudal de 3.5 L/min, una presión máxima de 0.48 MPa y una altura de succión de 1.5 m. Su construcción compacta y uso de motor

de corriente continua la hacen idónea para sistemas pequeños de agua o riego (Hi-Fi SAC, s.f.).

La Tabla 24 muestra las especificaciones técnicas de una Bomba de agua DP-521.

Tabla 25

Especificaciones de Bomba de agua DP-521

Características	Detalles
Voltaje de funcionamiento	12 V DC
Corriente Nominal	2 A
Caudal	3.5 L/min
Presión máxima	0.48 MPa (aprox. 4.8 bar)
Altura de succión	1.5 m.
Tipo	Diafragma, no sumergible

En la Figura 64, podemos observar un anemómetro de cazoletas.

Figura 19

Bomba de agua DP-521

Nota. Adaptado de *Bomba de agua DP-521* [Fotografía], por Hifisac (s.f.), en https://hifisac.com/shop/dp-521-bomba-de-agua-dp521-de-12vdc-y-2amperios-flujo-3-5l-min-no-sumergible-1681

3.3.10. PowerBank Redmi PB200LZM

El Redmi 20 000 mAh Fast Charge Power Bank es una batería portátil de polímero de litio de 3,7 V y 74 Wh, con dos puertos de entrada (Micro-USB y USB-C) y dos salidas USB-A que admiten carga rápida de hasta 18 W, incorporando protección contra sobrecarga, cortocircuito y sobrecalentamiento (Xiaomi Perú, s.f.).

La Tabla 25 muestra las especificaciones técnicas del PowerBank Redmi PB200LZM.

Tabla 26

Especificaciones técnicas del PowerBank Redmi PB200LZM.

Características	Detalles
Tipo de batería	Polímero de litio (Li-Po)
Capacidad	20 000 mAh / 74 Wh (3.7 V)
Capacidad nominal	12 000 mAh (5.1 V / 3.6 A)
Puertos de entrada	Micro-USB y USB-C
Parámetros de entrada	Hasta 18 W (5 V/2.1 A, 9 V/2.1 A o 12 V/1.5 A)
Puertos de salida	2 × USB-A
Parámetros de salida	5.1 V/2.4 A, 9 V/2 A, 12 V/1.5 A (≈18 W)
Dimensiones	154 × 73.6 × 27.3 mm
Protecciones incluidas	Contra sobrecarga, descarga excesiva, sobrecalentamiento, cortocircuito, sobretensión, etc.

En la Figura 65, se puede apreciar el powerbank Redmi Pb200LZM.

Figura 65

Power bank Redmi Pv200LZM

Nota. Adaptado de 20000 mAh Redmi Fast Charge Power Bank [Especificaciones], por Xiaomi Perú (s.f.), *en* https://www.mi.com/pe/product/20000mah-redmi-fast-charge-power-bank/specs

3.3.11. Aspersor de 360 grados

El aspersor es un dispositivo de riego rotativo de 360 grados que opera con una presión de trabajo entre 1.5 y 3 kg/cm², un caudal de 80 a 100 litros por hora y un radio de alcance de hasta 3 metros. Su diseño permite una distribución uniforme del agua en áreas pequeñas, lo que lo hace adecuado para sistemas de irrigación localizada (AliExpress, s.f.).

La Tabla 26 muestra las especificaciones técnicas de un aspersor de 360 grados.

Tabla 27

Especificaciones de los aspersores

Parámetros	Valores
Presión de trabajo	1.5 – 3 kg/cm²
Caudal	80 – 100 L/h
Radio de riego	0 – 3 m
Método de riego	Aspersión a 360°
Altura del microaspersor	54 mm (2.12 pulgadas)
Diámetro de entrada	4.5 mm (0.17 pulgadas)

En la Figura 66, podemos observar un aspersor de 360 grados.

Figura 66

Aspersor de 360 grados

Nota. Adaptado de *Aspersor de 360 grados* [Fotografía], por Aliexpress (s.f.), en https://es.aliexpress.com/item/1005006411484272.html

3.3.12. Manguera de polietileno

La manguera de polietileno es un conducto flexible, ligero y resistente, fabricado con polietileno de alta o baja densidad, ampliamente utilizado para transportar agua en sistemas de riego y conducción a baja presión. Destaca por su alta resistencia a la presión (hasta 25 bares), durabilidad frente a impactos y agentes químicos, así como por su fácil instalación y manipulación (Racsa Riego, s.f.).

La Tabla 27 muestra las especificaciones técnicas de la manguera de polietileno.

Tabla 28

Especificaciones de la manguera de polietileno

Propiedades	Valor / Descripción
Material	Polietileno de alta densidad (PE100)
Diámetro nominal	Desde 20 mm hasta 1 200 mm
Presión nominal	4 a 25 bares
Densidad	0,957–0,961 g/cm³
Resistencia química	Buena frente a productos comunes; atóxica
Resistencia al impacto	Elevada (flexibilidad frente a sismos y golpes)

En la Figura 67, podemos observar una manguera de polietileno.

Figura 67

Manguera de polietileno

Nota. Adaptado de *Manguera de polietileno negra* [Fotografía], por Racsa (s.f.), en https://www.racsariego.com/product/manguera-polietileno-negra-ced-40-de-1-1-2-100-m-diametro-de-38-mm/?v=ac761520fac3

3.3.13. Acrílico

El acrílico, o polimetilmetacrilato (PMMA), es un termoplástico transparente ampliamente utilizado por su alta transmisión de luz, resistencia al impacto y facilidad de mecanizado. Gracias a estas propiedades, se emplea frecuentemente en la fabricación de cubiertas, carcasas y componentes protectores para dispositivos electrónicos. Según Piedmont Plastics (2023), el acrílico ofrece una transmisión de luz superior al 92 %, una resistencia al impacto hasta 17 veces mayor que la del vidrio y un rango térmico de trabajo que va desde –40 °C hasta 80 °C.

La Tabla 28 muestra las especificaciones técnicas de una plancha de acrílico.

Tabla 29

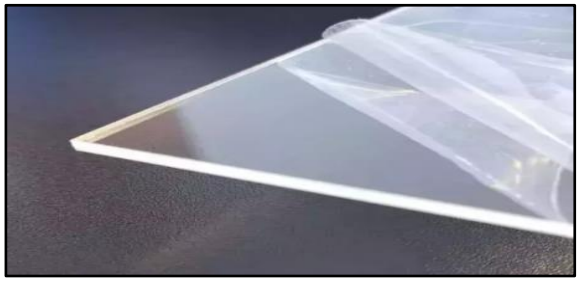
Especificaciones de acrílico

Propiedades	Valores
Densidad	~1,19 g/cm³
Transmisión de luz	>92 %
Resistencia al impacto	Hasta 17 veces más que el vidrio
Rango térmico de trabajo	–40 °C a +80 °C
Dureza (escala Rockwell)	M 102
Absorción de agua	~0,2 % en 24 horas
Resistencia química	Buena frente a productos comunes; sensible a solventes fuertes
Mecanizado	Compatible con corte láser, fresado y termoformado

En la Figura 68, podemos observar una plancha de acrílico transparente.

Figura 20

Lamina de acrílico



Nota. Adaptado de *Lámina de acrílico* [Fotografía], por Dojem (s.f.), en https://dojemmaterialesautorizado.mitiendanube.com/productos/acrilico-transparente-6mm/

3.4. Operacionalización de variables

La Tabla 29 presenta el esquema de operacionalización de las variables analizadas en la investigación.

Tabla 30

Operacionalización de variables

Variable	Definición operacional	Dimensión	Indicador
Humedad ambiental	Cantidad de vapor de agua presente en el aire, medida en porcentaje.	Condiciones ambientales	0–100 % HR
Temperatura del aire	Grado de calor del aire medido en °C, según principios termodinámicos.	Clima externo	-10 °C a 50 °C
Nivel de agua	Distancia entre el sensor y el agua, determinada por ondas ultrasónicas.	Disponibilidad de agua	Distancia en cm
Humedad del suelo	Porcentaje de agua en el sustrato detectado por variación capacitiva.	Estado del suelo	0–100 % humedad del suelo
Velocidad del viento	Rapidez del aire en movimiento, obtenida por rotación de cazoletas.	Condiciones ambientales	km/h
Temperatura infrarroja	Medición infrarroja de temperatura del cielo, útil para detectar nubosidad.	Radiación solar superficial	°C

3.5. Pruebas

En la Figura 69 se presenta el prototipo previo al montaje para las pruebas de funcionamiento.

Figura 69

Prototipo listo para las pruebas

Debido a la naturaleza experimental del prototipo y la dificultad de reproducir con exactitud las condiciones ambientales propias de una helada, se procedió a modificar los umbrales de activación del sistema con el fin de simular dichos escenarios y evaluar su comportamiento en un entorno controlado, como se muestra en la Tabla 30.

Tabla 31

Rangos de activación

Variable	Sensor	Unidad	Rango de activación real	Rango de activación controlado
Temperatura ambiente	DS18B20	°C	< 2°C	25 °C ≤
Temperatura del cielo	MLX90614	°C	< -10°C	< -10°C
Velocidad del viento	Anemómetro	km/h	< 5 km/h	< 2 km/h
Humedad del suelo	Sensor capacitivo V1.2	%	<30%	<30%
Humedad relativa	DHT22	%	< 60%	< 65%
Nivel de agua	Ultrasónico HC-SR04	cm	Depende de la profundidad del agua	>30 cm

3.5.1. Visualización de Lecturas y estados de los nodos sensores

Las lecturas y estados se pueden visualizar a través del dashboard de Node-Red como se muestra en la Figura 70 y 71.

Figura 70

Prueba de visualización de lecturas de sensores

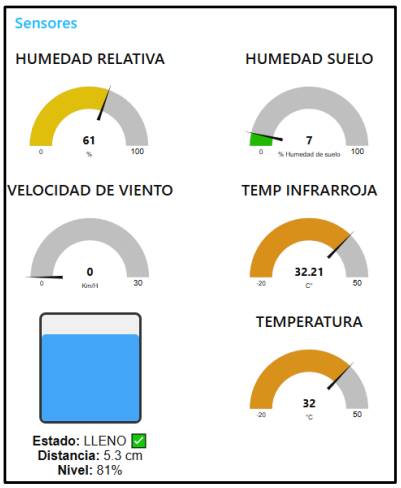
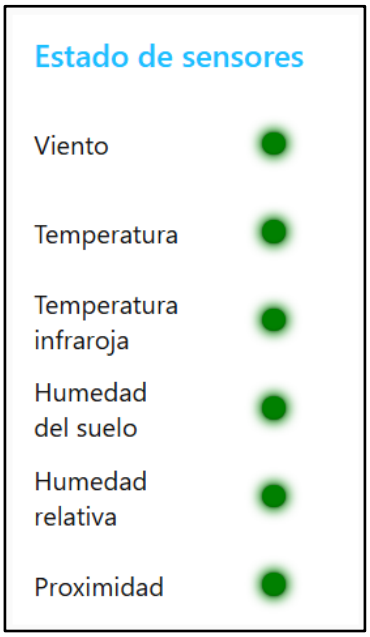


Figura 21

Visualización de estados de los Sensores



3.5.2. Visualización de lecturas almacenadas en la base de datos

Para la visualización del historial de lecturas se tiene que seleccionar la fecha deseada, así como también se puede visualizar las lecturas de las 2 últimas horas, como se puede apreciar en la Figura 72 y 73.

Figura 22

Visualización de estados de los Sensores por fecha

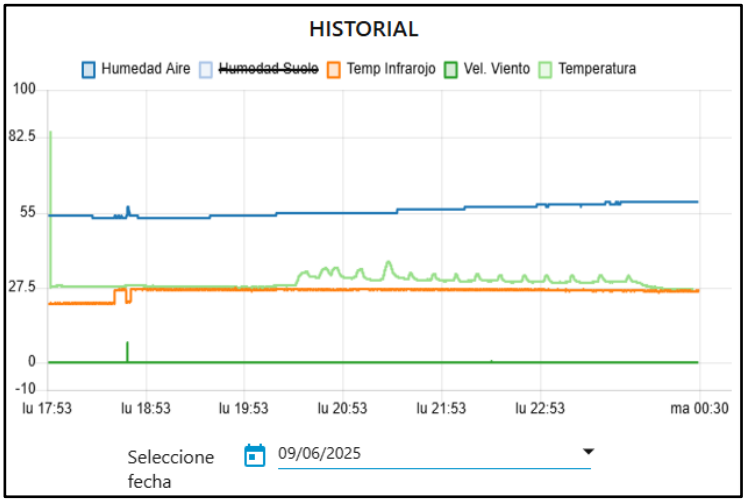
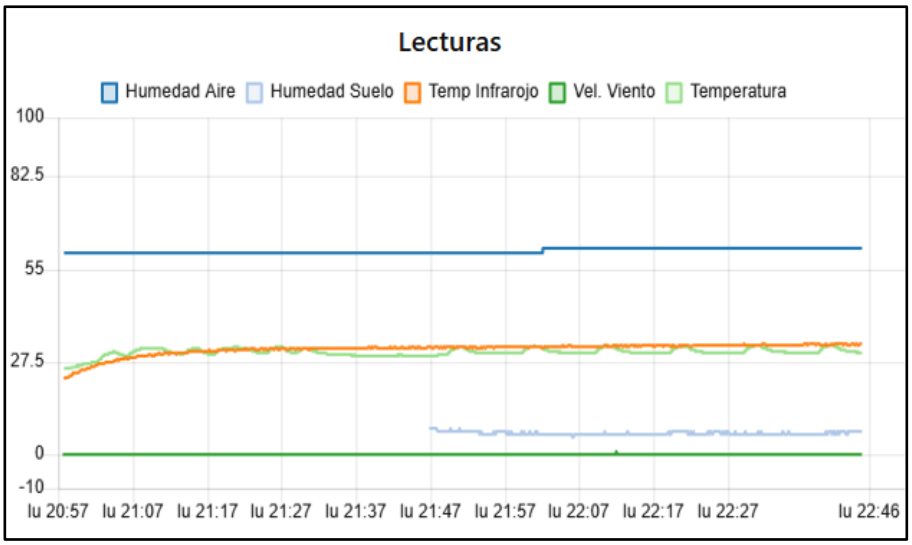


Figura 73

Visualización de lecturas de las últimas 2 horas



3.5.3. Activación de bomba de agua ante helada inminente y envío de notificación a Telegram

Cuando se cumplen las condiciones ante una helada inminente se manda la señal de activación a la bomba de agua y a su vez el envío de una notificación de alerta a Telegram.

En las Figuras 74 y 75 se pueden apreciar la activación de la bomba de agua y la notificación de Telegram.

Figura 23

Activación de bomba de agua ante helada inminente

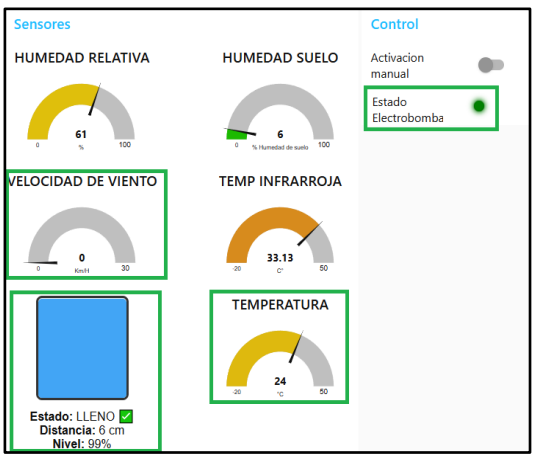
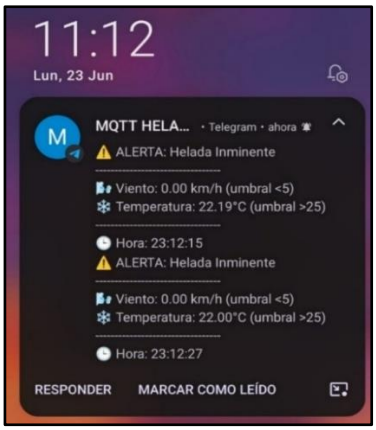


Figura 75

Notificación de Telegram ante helada inminente



3.5.4. Activación preventiva de bomba de agua ante posibles heladas inminente y envío de notificación a Telegram

Cuando se cumplen las condiciones ante posibles heladas se manda la señal de activación a la bomba de agua y a su vez el envío de una notificación de alerta a Telegram.

En las Figuras 76 y 77 se pueden apreciar la activación preventiva de la bomba de agua y la notificación de Telegram ante posibles heladas.

Figura 246

Activación preventiva de bomba de agua

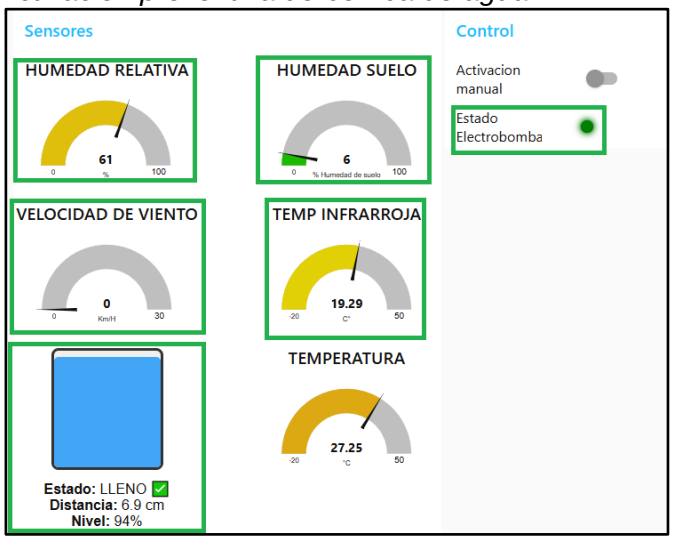


Figura 257

Notificación de Telegram ante posibles heladas

3.5.5. Comportamiento de los sensores en condiciones de temperaturas bajas

El prototipo fue probado en un ambiente frío con el objetivo de comprobar que los sensores funcionen adecuadamente durante episodios de heladas, como se puede apreciar en las Figuras 78, 79 y 80.

Figura 268

Funcionamiento del sensor DHT22 y DS18B20 en temperaturas bajo 0 ºC

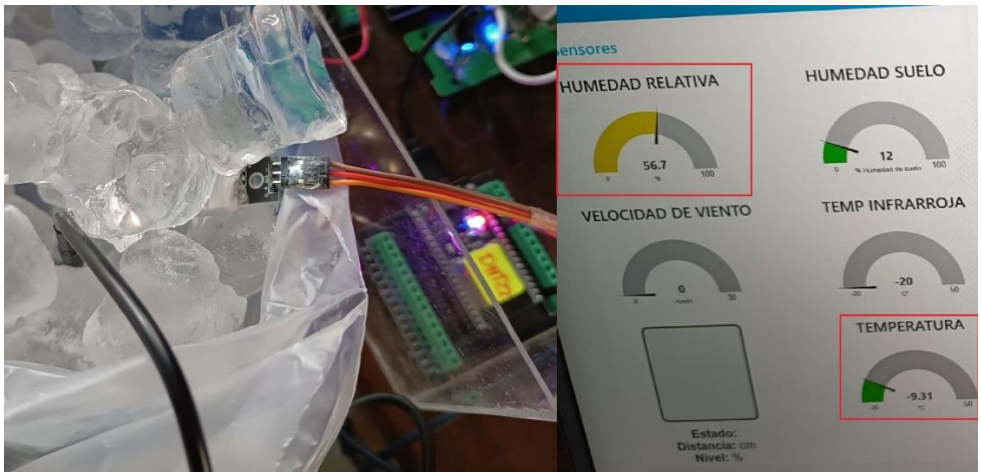


Figura 79

Funcionamiento del sensor de humedad capacitivo v1.2 en temperaturas bajo 0 ºC

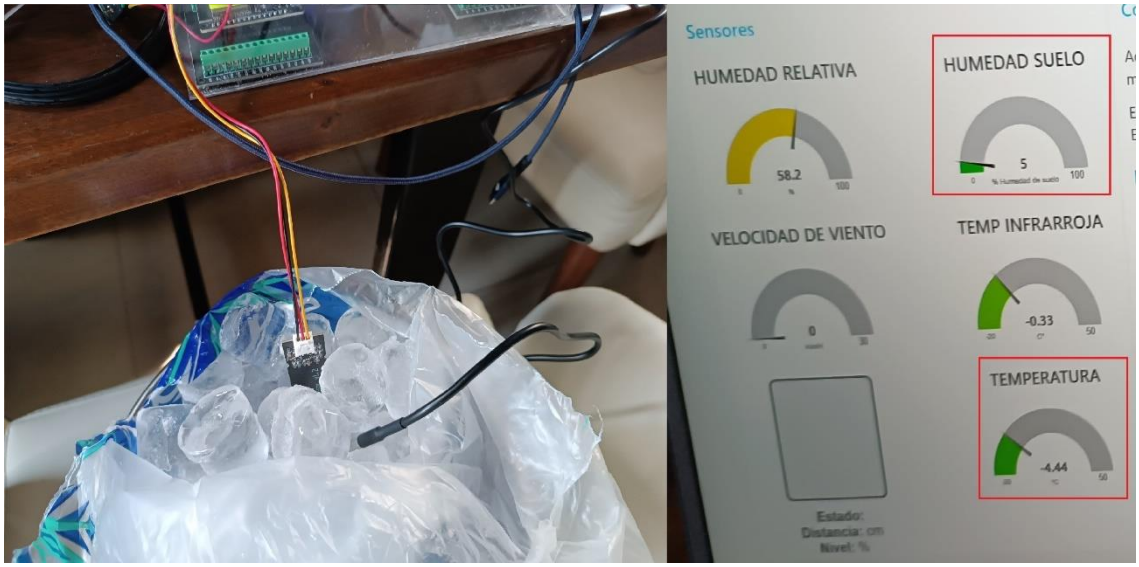
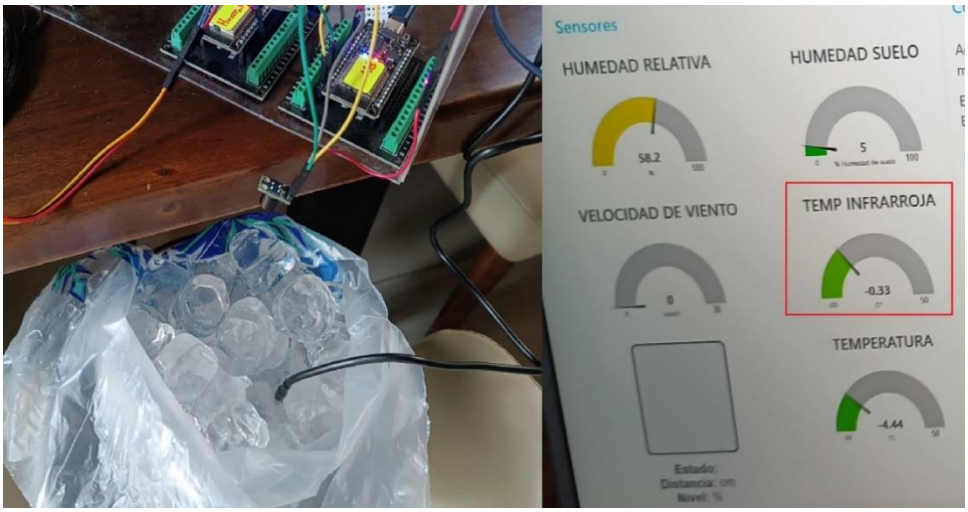


Figura 80

Funcionamiento del sensor infrarrojo MLX90614 en temperaturas bajo 0 ºC



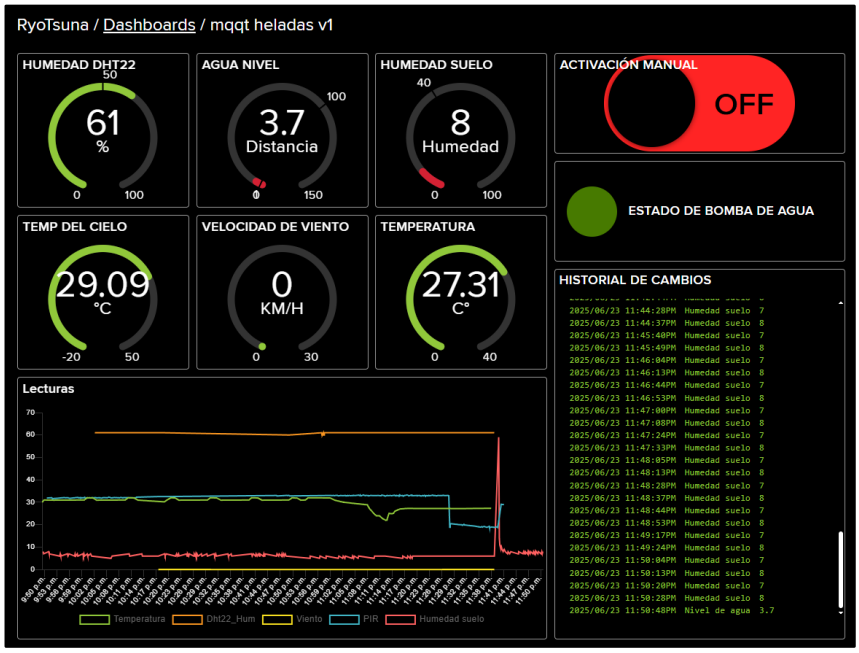
3.5.6. Monitorización mediante ADAFRUIT IO

Para la monitorización remota se ingresó a la plataforma ADAFRUIT IO, mediante el cual se pudo visualizar las lecturas de los sensores, historial y estado de la bomba de agua.

La Figura 81 muestra el interfaz de Adafruit IO de nuestro prototipo.

Figura 81

Monitorización en ADAFRUIT IO



CAPÍTULO IV: RESULTADOS

En este capítulo se presentan los resultados obtenidos a partir de las pruebas realizadas al prototipo. Durante el proceso experimental se simularon condiciones de helada y escenarios que podían anticipar una posible helada para observar la respuesta del prototipo. También se comprobó la **activación manual** del riego por aspersión para verificar el funcionamiento del actuador. Los datos registrados permitieron evaluar el desempeño del prototipo y confirmar que operaba conforme a las condiciones establecidas.

4.1. Resultados de la Prueba de Simulación de helada Inminente

Durante las simulaciones de helada, la temperatura ambiente fue disminuyendo de forma gradual para acercarla a condiciones reales de riesgo. El prototipo comprobó continuamente la velocidad del viento y el nivel de agua para asegurar que la aspersión funcione correctamente.

En la tabla 31, se puede observar las condiciones de activación del riego ante helada inminente.

Tabla 31

Condiciones de activación del riego ante helada inminente

Parámetro	Condición requerida	Acción del sistema
Velocidad del viento	< 5 km/h	Permite activar riego
Temperatura ambiente	≤ 2 °C	Activar riego / Envío de alerta
Nivel de agua del reservorio	≥ 10%	Permite activar riego
Estado del sensor principal	Operativo / sensor de respaldo	Usar sensor de respaldo si el sensor principal falla
Apagado del sistema	T° > 5 °C	Desactiva riego

En la tabla 32, se puede observar los valores de registro de temperatura durante la prueba.

Tabla 32

Registro de temperatura durante la prueba

Tiempo (min)	Temp. ambiente (°C)	Estado del sistema
0	20	*Off*
5	16	*Off*
10	10	*Off*
15	6	*Off*
20	2	*On – Riego activado*
25	-3	*On*
30	-5	*On*

El sistema activó el riego automáticamente cuando la temperatura descendió por debajo del umbral establecido (2 °C). El riego permaneció encendido mientras la temperatura siguió disminuyendo.

4.2. Resultados de la Prueba de Activación Preventiva por Posible Helada

Se simularon condiciones de ambiente típicas de noches despejadas donde existe probabilidad de que ocurra una helada. En estas pruebas se analizaron la humedad del suelo, la humedad relativa del ambiente y la temperatura infrarroja del cielo.

En la tabla 33, se puede observar los valores de condición de activación preventiva ante una posible helada.

Tabla 33

Condiciones de activación preventiva

Parámetro	Condición requerida	Acción del sistema
Velocidad del viento	< 5 km/h	Permite activar riego
Humedad relativa	< 60%	Indica riesgo de helada
Temp. IR del cielo	< –10 °C	Indica cielo despejado
Humedad del suelo	< 35%	Activar riego preventivo/ Envío de alerta
Apagado del sistema	Humedad del suelo ≥ 35%	Detiene riego

En la tabla 34, se puede observar los valores de registro de humedad y temperatura del cielo durante la prueba realizada.

Tabla 34

Registro de humedad y temperatura del cielo durante la prueba

Tiempo (min)	Humedad del suelo (%)	Humedad relativa (%)	Temp. IR cielo (°C)	Acción
0	47	55	-6	Sin riego
5	42	52	-8	Sin riego
10	34	59	-11	*Riego preventivo On*
15	33	58	-15	On
20	35	59	-13	Off (suelo Húmedo)

El sistema activó el riego de manera preventiva cuando la humedad del suelo bajó a 34% y la temperatura infrarroja del cielo indicó cielo despejado (-11 °C). El riego se detuvo cuando la humedad del suelo llegó nuevamente al 35%.

4.3. Resultados de Velocidad de Viento

Durante las pruebas se verificó cómo reaccionaba el prototipo ante velocidades de viento elevadas.

En la tabla 35, se puede observar los valores de medición de velocidad de viento.

Tabla 35

Medición de velocidad del viento

Tiempo (min)	Vel. viento (km/h)	Acción del sistema
0	4	Normal
10	4	Normal
20	3	Normal
25	4	Normal
30	6	*No activa riego*

Cuando el viento superó los 5 km/h, el sistema bloqueó la activación del riego, tal como estaba configurado.

4.4. Resultados de Nivel de Agua del Reservorio

En la tabla 36, se registraron los niveles de agua para verificar el estado del reservorio.

Tabla 36

Nivel de agua del reservorio

Nivel de agua (%)	Acción
100%	Riego permitido
80%	Riego permitido
60%	Riego permitido
30%	*Alerta enviada (nivel de agua bajo)*
0%	*Riego bloqueado*

4.5. Activación Manual del Sistema de Riego

Se implementaron dos switches para la activación manual del sistema de riego: uno en el panel de control local de Node-RED, que permite encender o apagar el sistema desde la red local, y otro en la plataforma Adafruit IO, el cual permite realizar el mismo control de forma remota a través de internet. Esto garantiza la operatividad del sistema tanto en campo como a distancia, sin depender de las condiciones automáticas definidas por los sensores

En las Figuras 82 y 83 se muestra la activación manual de la bomba de agua mediante el *dashboard* de Node-RED y Adafruit IO.

Figura 27

Activación manual NODE-RED

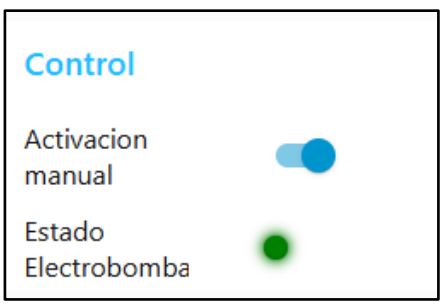
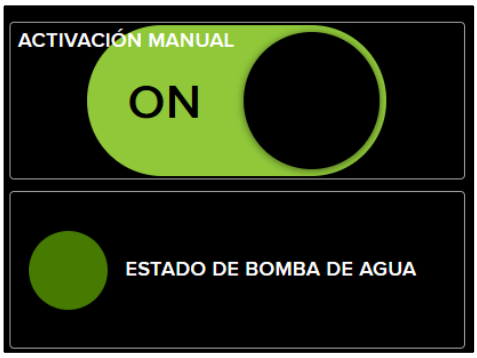


Figura 283

Activación manual ADAFRUIT IO



Las Figuras 84 y 85 muestran los aspersores en funcionamiento, así como su efectividad para mantener las plantas húmedas, lo cual es fundamental para protegerlas de las heladas.

Figura 294

Aspersor funcionando con la activación manual

Figura 305

Plantas regadas con el aspersor

La figura 86 muestra el montaje final del prototipo.

Figura 316

Armado final del prototipo

CAPÍTULO V: DISCUSIÓN

El prototipo desarrollado en esta investigación demostró ser eficaz en la detección de condiciones de helada y en la activación automatizada del riego por aspersión, con una arquitectura basada en módulos ESP32, protocolo MQTT, visualización de parámetros mediante Node-RED, Adafruit IO y envió de notificaciones a TELEGRAM.

En comparación con lo reportado por Arhancet (2024), quien evaluó tres sistemas de control de heladas en cultivos de cerezos, se observa una coincidencia en cuanto a la aplicación del riego por aspersión como estrategia de mitigación ante heladas, aunque el enfoque de este autor no se basó en la integración de tecnologías IoT. Esto resalta el valor añadido del presente prototipo, que permite un monitoreo remoto y en tiempo real, optimizando recursos.

De forma similar, Castro (2023) implementó una solución IoT en un invernadero inteligente, donde se utilizaron sensores de temperatura y humedad para preservar cultivos de tomates. A diferencia de su trabajo, que se basó en plataformas como REACT, la presente propuesta optó por MQTT, Node-RED, ADAFRUIT IO Y TELEGRAM lo que permitió una mayor personalización del dashboard y menor latencia en la comunicación.

El estudio de Cevallos y Mosquera (2022) utilizó ESP32 para monitorear cultivos de arroz mediante ThingSpeak. Aunque se comparten tecnologías similares, nuestro prototipo presenta un enfoque diferente en cuanto a la lógica de control, emplea el protocolo MQTT y Node-RED para ejecutar acciones localmente, con menor latencia y mayor control ante condiciones críticas como las heladas. Esta integración permite no solo monitorear, sino también responder en tiempo real, representando un avance funcional frente a modelos más dependientes de plataformas externas.

Finalmente, en cuanto al hardware utilizado, los sensores seleccionados demostraron un desempeño confiable en entornos fríos, validando las especificaciones técnicas descritas por Analog Devices, SparkFun y Naylamp Mechatronics. Esta observación se alinea con el objetivo de esta tesis de garantizar la funcionalidad del sistema incluso en condiciones de baja temperatura.

CONCLUSIONES

Respecto a los efectos de las heladas, no se realizaron pruebas en campo debido a limitaciones climáticas y de entorno. Sin embargo, según la bibliografía revisada, las heladas provocan la formación de hielo intracelular, lo que genera deshidratación, ruptura de membranas celulares y muerte de tejidos vegetales, ocasionando daños severos en los cultivos. Además, se recomienda el riego por aspersión como un método activo de protección.

Se logró identificar y establecer los parámetros críticos que afectan a los cultivos durante eventos de heladas, como la temperatura ambiente, la humedad del suelo, la velocidad del viento y la radiación solar. Asimismo, se definieron los parámetros esenciales de hardware, incluyendo sensores específicos (DS18B20, DHT22, anemómetro, etc.) y actuadores como electroválvulas, así como los componentes de software necesarios, como el protocolo MQTT y la plataforma Node-RED. Esta integración permitió desarrollar un prototipo capaz de monitorear y actuar de forma eficiente ante condiciones climáticas adversas, sentando las bases para su aplicación en entornos agrícolas vulnerables a las heladas.

La arquitectura basada en ESP32 con comunicación WiFi vía MQTT resultó adecuada para prototipos de corto alcance y bajo costo, por sus ventajas en consumo energético, baja latencia y fiabilidad en la transmisión de datos. Además, su integración con plataformas IoT como Adafruit IO facilita el monitoreo en la nube y la toma de decisiones en tiempo real, lo que la hace eficiente para proyectos agrícolas localizados.

Aunque el sistema no fue probado en un entorno real de helada, su diseño y funcionamiento permiten establecer una base técnica sólida para futuras validaciones en campo.

RECOMENDACIONES

Se recomienda realizar mediciones en zonas altoandinas para validar de manera directa los efectos de las bajas temperaturas en cultivos altoandinos y complementar así la revisión teórica presentada en este trabajo.

Asimismo, se sugiere implementar el prototipo en campos agrícolas altoandinos para evaluar su funcionamiento en condiciones más exigentes, verificar su resistencia y determinar su efectividad frente a eventos de helada, además de analizar su impacto en la protección del cultivo y en el uso eficiente del agua.

También se sugiere continuar con las pruebas experimentales para afinar los parámetros de activación y mejorar la estabilidad del sistema. Realizar más ciclos de prueba permitirá obtener datos adicionales y asegurar que la activación del riego sea precisa y oportuna ante cambios bruscos de temperatura.

Además, se recomienda reforzar la instalación física del prototipo, incorporando cajas protectoras para los sensores y el microcontrolador, a fin de garantizar su resistencia en ambientes rurales y condiciones climáticas adversas. Del mismo modo, se recomienda mantener el sistema alimentado con energía solar y baterías para asegurar su funcionamiento continuo en zonas donde el acceso a la red eléctrica es limitado.

Por último, se aconseja continuar registrando y analizando los datos obtenidos durante las pruebas, ya que esto permitirá mejorar la toma de decisiones y facilitar la evolución del sistema en futuras implementaciones.

REFERENCIAS BIBLIOGRÁFICAS

Adafruit Industries. (s.f.). Adafruit IO. Recuperado de https://io.adafruit.com/

Analog Devices. (s.f.). DS18B20 Programmable Resolution 1-Wire Digital Thermometer. https://www.analog.com/en/products/ds18b20.html?form=MG0AV3

Arhancet, J. (2024). *Evaluación de tres sistemas de control de heladas mediante riego por aspersión en cultivo de cerezos (prunus avium L.) en el valle de Los Antiguos, provincia de Santa Cruz* [Tesis de Maestría, Universidad Nacional de Cuyo]. https://bdigital.uncu.edu.ar/objetos_digitales/20386/tesis-arhancet-mrd-2024-aprobada-enviar-uncuyo.pdf

Biblioteca Digital INIA. (2020). *Riego por aspersión*. Instituto Nacional de Innovación Agraria. https://biblioteca.inia.cl/server/api/core/bitstreams/0340e3be-f94f-4fc1-92d2-e143055740ba/content

Castro P. (2023). *Smart greenhouse aplicando las IoT para la conservación del cultivo de tomates nativos en la región Lambayeque* [Tesis de Pregrado, Universidad Católica Santo Toribio de Mogrovejo]. https://tesis.usat.edu.pe/bitstream/20.500.12423/7186/1/TL_CastroFernandezPaola.pdf

Cevallos y Mosquera (2022). *Diseño e implementación de un prototipo IoT para el monitoreo de parámetros ambientales aplicados al cultivo de arroz utilizando ESP32 y Thingspeak* [Tesis de Pregrado, Universidad Politécnica Salesiana]. https://dspace.ups.edu.ec/bitstream/123456789/22884/4/UPS-GT003870.pdf

Electronic Lab. (s.f.). *Board de desarrollo con ESP32 WiFi + Bluetooth BLE CH9102F – 38 pines*. Recuperado de https://electronilab.co/tienda/board-de-desarrollo-con-esp32-wifi-bluetooth-ble-ch9102f-38-pines/

Electropeak (2022). *Aosong DHT22 Temperature Humidity Sensor.* https://electropeak.com/sensor-dht22-1?srsltid=AfmBOooxFA84KOkQqEvFM7gRjsqM2Hl6tjjhpZt-hdjFy7d5kLv0wxrl

Espinoza M. (2017). *Análisis transcriptómico de la respuesta a heladas en papas nativas.* [Tesis de Maestría, Universidad Peruana Cayetano Heredia]. https://repositorio.upch.edu.pe/bitstream/handle/20.500.12866/1037/Analisis_EspinozaKou_Mey-Ling.pdf;jsessionid=512E8B26B5EDC8CC5C6E76163216BB6B?sequence=3

Espressif Systems. (2025). *ESP32 Series datasheet* (Power Consumption by Power Modes). Recuperado de https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

FAO (2010). *Protección contra las heladas: fundamentos, práctica y economía* [Libro PDF, Organización de las Naciones Unidas para la Agricultura y la Alimentación]. https://www.fao.org/4/y7223s/y7223s.pdf

Go Daddy (2024). *Raspberry Pi: Qué es, cómo funciona y proyectos que puedes realizar*. Recuperado de https://www.godaddy.com/resources/es/tecnologia/que-es-raspberry-pi

Go Daddy. (2024). *Telegram: qué es, cómo funciona y por qué usarlo como app de mensajería*. Recuperado de https://www.godaddy.com/resources/latam/general/que-es-telegram

Halfacree, G. (2018). *The official Raspberry Pi beginner's guide: How to use your new computer. Raspberry Pi Press*. https://magazines-attachments.raspberrypi.org/books/full_pdfs/000/000/038/original/BeginnersGuide-4thEd-Eng_v2.pdf

Hifi SAC (s.f.) *Bomba de Agua DP521 de 12VDC y 2Amperios*. Flujo 3.5L/min (No Sumergible). Recuperado de https://hifisac.com/shop/dp-521-bomba-de-agua-dp521-de-12vdc-y-2amperios-flujo-3-5l-min-no-sumergible-1681#attr=

Huanca V. (2024). *Implementación de un sistema inalámbrico utilizando el internet de las cosas para el monitoreo del cultivo de papas en el fundo Huaylacucho en el departamento de Ayacucho* [Tesis de pregrado, Universidad de Ciencias y Humanidades]. https://repositorio.uch.edu.pe/bitstream/handle/20.500.12872/986/Huanca_V_tesis_ingenieria_electronica_telecomunicaciones_2024.pdf?sequence=3&isAllowed=y

Kurniawan, A. (2019). *Internet of Thing Projects with ESP32* https://else.fcim.utm.md/pluginfile.php/90372/mod_resource/content/1/esp32%20programming.pdf

Mahedero F. (2020). *Desarrollo de una aplicación IoT para el envío de imágenes mediante el protocolo MQTT* [Tesis de Pregrado, Universidad Politécnica de Valencia]. https://riunet.upv.es/bitstream/handle/10251/152408/Mahedero%20-%20Desarrollo%20de%20una%20aplicaci%c3%b3n%20IoT%20para%20el%2

0env%c3%ado%20de%20im%c3%a1genes%20mediante%20el%20protocolo%20MQTT..pdf?sequence=1&isAllowed=y

Naylamp Mechatronics (s.f.) *Sensor de humedad de suelo capacitivo V1.2.* https://naylampmechatronics.com/sensores-temperatura-y-humedad/538-sensor-de-humedad-de-suelo-capacitivo-v1.html

Naylamp Mechatronics. (s.f.). *Sensor de temperatura infrarrojo MLX90614 GY-906.* https://naylampmechatronics.com/sensores-temperatura-y-humedad/330-sensor-de-temperatura-infrarrojo-mlx90614-gy-906.html

Organización Meteorológica Mundial. (1992). *International Meteorological Vocabulary (2nd ed.).* OMM-No. 182. https://library.wmo.int/records/item/35809-international-meteorological-vocabulary?offset=2

Pandora FMS (2025). *¿Qué es MQTT? El protocolo más utilizado para IoT.* https://pandorafms.com/es/it-topics/que-es-mqtt/

Piedmont Plastics (s.f.) *The technical properties of acrylic sheet: A comprehensive guide.* https://www.piedmontplastics.com/blog/acrylic-sheet-properties?srsltid=AfmBOorxwSj2Koi0YpqeVWXbkppl8WMTwn1rVFj3QGZeUCBsj78HDNwM&utm_source=%20%20%20%20acrilico

Proaño L. (2024). *Implementación de un prototipo de un sistema de monitoreo de cantidad de luz, humedad del suelo y temperatura, para cultivo de plantas, basado en ESP32 y página web* [Tesis de Pregrado, Escuela Politécnica Nacional]. https://bibdigital.epn.edu.ec/bitstream/15000/25596/1/CD%2014127.pdf

Profesional Review (Solé R.). *Raspberry Pi: Crea proyectos DIY por muy poco dinero.* https://www.profesionalreview.com/2021/07/18/que-es-raspberry-pi/

Random Nerd Tutorials. (s.f.). *Complete guide for ultrasonic sensor HC-SR04*. Recuperado de https://randomnerdtutorials.com/complete-guide-for-ultrasonic-sensor-hc-sr04/?form=MG0AV3

Rechner. (2022). *Sensores de temperatura industriales*. Rechner Sensors. https://www.rechner-sensors.com/es/documentacion/knowledge/el-sensor-de-temperatura

Red Hat. (2023). *¿Qué es Internet de las cosas (IoT)?* Recuperado de https://www.redhat.com/es/topics/internet-of-things

Rubiños S. (2024). *Monitoreo de la calidad del agua mediante tecnología lora en un sistema hidropónico para mejorar el cultivo de lechuga en zonas urbanas de Ventanilla, Callao 2024* [Informe Final de Investigación, Universidad Nacional del Callao]. https://repositorio.unac.edu.pe/bitstream/handle/20.500.12952/9655/informe%20-%20rubi%c3%91os-damas.pdf?sequence=1&isallowed=y

SENAMHI. (2010). *Atlas de heladas del Perú. Servicio Nacional de Meteorología e Hidrología del Perú.* https://idesep.senamhi.gob.pe/portalidesep/files/tematica/atlas/helada/atlasheladas.pdf

Sensovant (s.f.). *Anemómetro de cazoletas.* https://sensovant.com/productos/meteorologia/sensores-viento-mecanicos/anemometro-de-cazoletas/

Sertec Riego (2021) *Sistema de riego por aspersión, una protección ante las bajas temperaturas*. Sertec Riego. https://sertecriego.com/sistema-de-riego-por-aspersion-una-proteccion-ante-las-bajas-temperaturas/

SparkFun Electronics. (s.f.). *DHT22 Temperature-Humidity Sensor Datasheet.* Recuperado de https://cdn.sparkfun.com/assets/f/7/d/9/c/DHT22.pdf

Stackhero (2022). *InfluxDB: Introducción*. Recuperado de https://www.stackhero.io/es-ES/services/InfluxDB/documentations/Introduccion

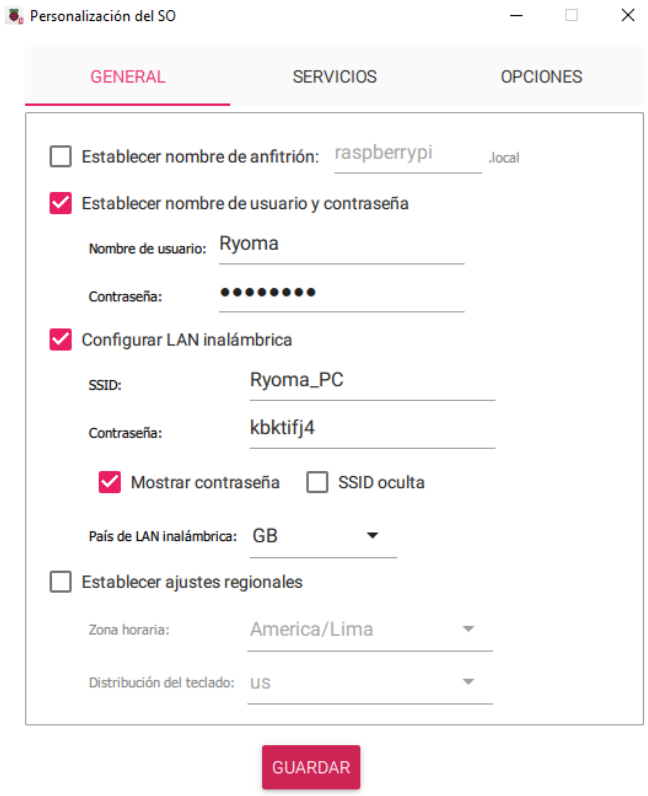
UNAM. (s.f.). *Microcontroladores y sistemas embebidos*. [Trabajo de Pregrado, Universidad nacional Autónoma de México]. https://www.cliffsnotes.com/study-notes/14809840

ANEXOS

Anexo 1. Matriz de consistencia

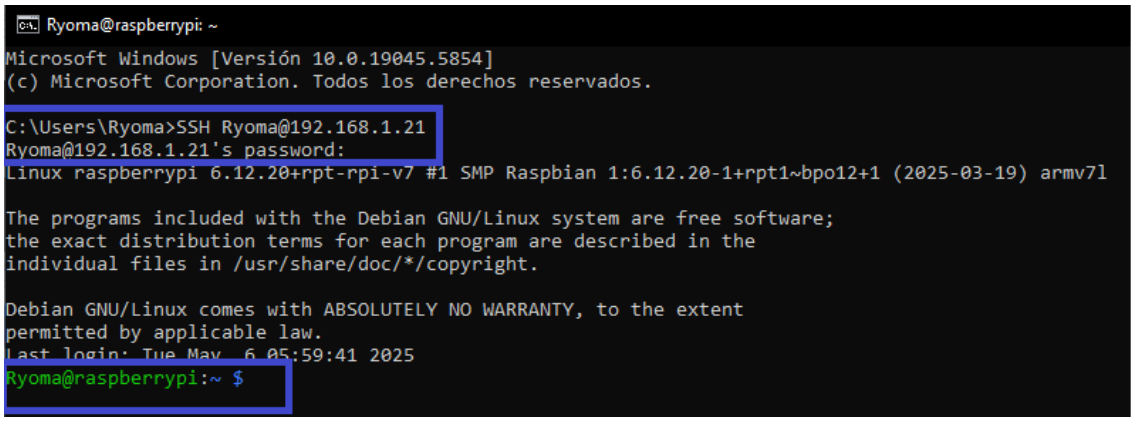
Problema	Objetivos	Variables	Indicador	Metodología
Problema general ¿En qué medida el uso de riego automatizado reducirá los efectos de las heladas en las zonas alto andinas? **Problemas específicos** ● ¿Cuáles son los efectos que producen las heladas en los cultivos? ● ¿Cuáles son los parámetros a controlar en los cultivos como producto de las heladas para el diseño de prototipo? ● ¿Qué tecnología de IOT es la más adecuada para el monitoreo remoto del prototipo?	**Objetivo general** Diseñar e implementar un prototipo de monitoreo de riego por aspersión que reduzca los efectos de las heladas en los cultivos de las zonas alto andinas del Perú. **Objetivos específicos** ● Determinar cuáles son los efectos de las temperaturas bajas producto de las heladas en los cultivos en las zonas alto andinas. ● Determinar los parámetros a controlar en los cultivos como producto de las heladas los parámetros de hardware y software del prototipo. ● Determinar qué tecnológica IoT es las más adecuada para el monitoreo remoto del prototipo	**Variable de estudio** Diseño e implementación de un sistema de riego por aspersión para reducir los efectos de las heladas en cultivos, monitoreado local y remotamente	Humedad relativa y del suelo Temperatura Viento Temperatura Nivel de agua	Medición de la humedad del aire y del suelo Medición de la temperatura a nivel del suelo Medición de la velocidad de viento Medición de la temperatura atmosférica Mide los niveles de agua disponible **Nivel de investigación:** - Aplicativo **Diseño de investigación** -Aplicativo

Anexo 2. Instalación y configuración del S.O. Raspberry Pi



Una vez encendido la Rasberry pi 3+ se procede a acceder a esta mediante el comando “SSH usuario@ (IP Raspberry)” y colocar la contraseña previamente establecida al momento de instalar el sistema operativo, para finalmente poder ingresar a la terminal de la Rasberry pi 3+.

Anexo 3. Configuración de Raspberry Pi 3+ mediante comando SSH



Anexo 4. Condiciones de activación preventiva ante posibles heladas

Ante la detección de una posible helada, el sistema de riego por aspersión se activa de forma preventiva. Este riego se mantiene activo hasta que la humedad del suelo supere el 35 %, ya que un suelo húmedo permite conservar mejor el calor acumulado durante el día y libera calor latente al congelarse, ayudando así a elevar o mantener la temperatura del suelo durante la noche. Para lo cual se tiene que cumplir las siguientes condiciones:

- Viento menor a 5 km/h
- Humedad menor al 60%
- Temperatura infrarroja del cielo menor a -10 °C
- Humedad de suelo menor a 35%

Se configuró un nodo Function en el cual se encarga de recibir y registrar las lecturas de los nodos MQTT In

Anexo 5. Código de registro de lecturas de nodos MQTT In

```
// === Guardar datos entrantes ===
switch (msg.topic) {
    case "PIR":
        context.set("temp_cielo", parseFloat(msg.payload));
        return null;
    case "ANEMOMETRO":
        context.set("viento", parseFloat(msg.payload));
        return null;
    case "SENSOR_DHT22":
        context.set("humedad_relativa", parseFloat(msg.payloa
        return null;
    case "PROXIMIDAD":
        context.set("distancia", parseFloat(msg.payload));
        return null;
    case "SENSOR_HUMEDAD":
        context.set("humedad_suelo", parseFloat(msg.payload))
        break;
    default:
        return null;
}
```

Recupera todos los valores guardados y el estado actual del sistema de riego además de la verificación de los datos.

Anexo 6. Recuperación y verificación de datos de nodos MQTT In

```
// === Obtener datos guardados ===
let temp_cielo = context.get("temp_cielo");
let viento = context.get("viento");
let humedad_suelo = context.get("humedad_suelo");
let humedad_relativa = context.get("humedad_relativa");
let distancia = context.get("distancia");
let estadoRiego = context.get("estadoRiego") || "OFF";

// === Validar que todos los datos existan ===
if (
    temp_cielo === undefined ||
    viento === undefined ||
    humedad_suelo === undefined ||
    humedad_relativa === undefined ||
    distancia === undefined
) {
    return null;
```

Si la distancia medida por el sensor ultrasónico es alta indicando tanque bajo o vacío, se apaga el sistema si estaba encendido, ya que no hay agua suficiente para el riego.

Anexo 7. Código de verificación de nivel de agua

```
// === Evaluar si el tanque está vacío ===
if (distancia >= 90) {
    if (estadoRiego === "ON") {
        context.set("estadoRiego", "OFF");
        msg.payload = "OFF";
        return msg;
    }
    return null;
}
```

Se evalúa las condiciones ante una posible helada para la activación de riego, así también para su desactivación si la humedad de suelo supera el valor de 35%

Anexo 8. Código para evaluar las condiciones y desactivar el sistema

```
// === Evaluar condiciones de activación del riego preventivo
let condicionesActivacion = (
    temp_cielo < -10 &&
    viento < 5 &&
    humedad_suelo < 35 &&
    humedad_relativa < 60 &&
    distancia < 30
);

// === Activar riego ===
if (condicionesActivacion && estadoRiego === "OFF") {
    context.set("estadoRiego", "ON");
    msg.payload = "ON";
    return msg;
}

// === Desactivar riego si se alcanzó humedad adecuada ===
if (estadoRiego === "ON" && humedad_suelo >= 35) {
    context.set("estadoRiego", "OFF");
    msg.payload = "OFF";
    return msg;
}
```

Anexo 9. Código para enviar notificaciones sobre alertas de heladas

Ante una helada inminente se configura un nodo función el cual se encarga de enviar la notificación a un nodo Telegram Sender, el cual finalmente envía la notificación a Telegrama. El nodo función recibe y almacena los datos de los sensores para luego leer los valores recibidos

```
// 1. Actualizar datos si corresponde
if (msg.topic === "ANEMOMETRO") flow.set('velocidadViento', Number(msg.payloa
if (msg.topic === "TEMPERATURA") flow.set('temp', Number(msg.payload));

// 2. Obtener datos actuales del contexto
const velocidadViento = flow.get('velocidadViento');
const temp = flow.get('temp');

// 3. Evaluar solo si ambos datos están presentes
if (
    typeof velocidadViento === 'number' &&
    typeof temp === 'number'
```

Evalúa las condiciones de helada y, si se cumplen, envía notificaciones a los usuarios mediante sus respectivos Id, la alerta es enviada cada 15 min.

```
    // ALERTA: solo si temperatura < 25 y viento < 5
    if (temp < 25 && velocidadViento < 5) {
        const lastAlert = flow.get('lastAlert') || 0;
        const now = Date.now();

        if (now - lastAlert >= 900000) {
            const mensaje = `
⚠ *ALERTA: Helada Inminente*
--------------------------------
*Viento:* ${velocidadViento.toFixed(2)} km/h _(umbral <5)_
*Temperatura:* ${temp.toFixed(2)}°C _(umbral >25)_
--------------------------------
*Hora:* ${new Date().toLocaleTimeString()}
            `;

            [5000496329, 1334046629].forEach(chatId => {
                node.send({
                    payload: {
                        chatId,
                        type: "message",
                        content: mensaje,
                        options: { parse_mode: "Markdown" }
                    }
                });
            });

            flow.set('lastAlert', now);
        }
    }
}

return null;
```

Para las alertas ante posibles heladas se configuro otro nodo Function el cual enviara las notificaciones al nodo Telegram Sender

Se almacena los datos más recientes de tres sensores, velocidad del viento, humedad relativa y temperatura infrarroja. Luego, el nodo recupera los valores guardados para utilizarlos en el análisis

Anexo 10. Código de almacenamiento de lectura de los sensores

```
// 1. Actualizar SIEMPRE los valores con los datos más recientes
if (msg.topic === "ANEMOMETRO") flow.set('velocidadViento', Number(msg.payload));
if (msg.topic === "SENSOR_DHT22") flow.set('humedad', Number(msg.payload));
if (msg.topic === "PIR") flow.set('tempInfrarroja', Number(msg.payload));

// 2. Obtener los valores más actuales del contexto
const velocidadViento = Number(flow.get('velocidadViento'));
const humedad = Number(flow.get('humedad'));
const tempInfrarroja = Number(flow.get('tempInfrarroja'));
```

El sistema evalúa si los valores cumplen tres criterios críticos que podrían indicar una helada por radiación:

- Viento menor a 5 km/h
- Humedad menor al 60%
- Temperatura infrarroja del cielo menor a -10 °C

La notificación será enviada cada 30 minutos mientras se sigan cumpliendo las condiciones establecidas. Estas notificaciones se remitirán a los mismos usuarios previamente mencionados, mediante sus respectivos identificadores de Telegram.

Anexo 11. Código para evaluar las condiciones para el envío de notificaciones

```
// 4. Solo evaluar si tenemos los 3 valores
if (velocidadViento !== null && humedad !== null && tempInfrarroja !== null) {
    if (velocidadViento < 5 && humedad < 55 && tempInfrarroja < 25) {
        const lastAlert = flow.get('lastAlert') || 0;
        const now = Date.now();

        if (now - lastAlert >= 10000) { // Esperar 10 segundos entre alertas
            const mensaje = `
⚠️ *ALERTA: Posible Helada por radiación*
---------------------------------
💨 *Viento:* ${velocidadViento.toFixed(2)} km/h _(umbral <5)_
💧 *Humedad:* ${humedad.toFixed(2)}% _(umbral <55)_
❄️ *Temp. superficie:* ${tempInfrarroja.toFixed(2)}°C _(umbral <25)_
```

Anexo 12. Código para la configuración del nodo sensor velocidad de viento

```
#include <WiFi.h>            // Librería para conectar el ESP32 al WiFi
#include <PubSubClient.h>    // Librería para usar el protocolo MQTT

// ===== CONFIGURACIÓN DE LA RED WiFi =====
const char* ssid = "rolando 2.4G";        // Nombre de la red WiFi
const char* password = "13121989";        // Contraseña del WiFi

// Dirección IP fija para el ESP32 y parámetros de red
IPAddress local_ip(192, 168, 1, 25);      // IP que usará el ESP32
IPAddress gateway(192, 168, 1, 1);        // Dirección del router
IPAddress subnet(255, 255, 255, 0);       // Máscara de subred
IPAddress primaryDNS(8, 8, 8, 8);         // DNS primario (Google)
IPAddress secondaryDNS(8, 8, 4, 4);       // DNS secundario

// ===== CONFIGURACIÓN DEL SERVIDOR MQTT =====
const char* mqtt_server = "192.168.1.21"; // IP del servidor MQTT (Mosquitto)
const int mqtt_port = 1883;               // Puerto estándar MQTT
const char* mqtt_topic = "ANEMOMETRO";    // Tema donde se publica la velocidad del viento
const char* status_topic = "estado/ANEMOMETRO"; // Tema donde se publica el estado del dispositivo
const char* clientID = "ANEMOMETRO_ESP32";      // ID único del cliente MQTT (este ESP32)

// ===== CONFIGURACIÓN DEL SENSOR =====
const int sensorPin = 32;     // Pin donde está conectado el sensor de viento
const float vRef = 3.3;       // Voltaje de referencia para conversión ADC
const int adcMax = 4095;      // Resolución máxima del conversor analógico-digital (12 bits)
const int reportInterval = 7000;  // Intervalo para enviar datos (7 segundos)

// ===== LED DE ESTADO =====
const int ledPin = 2;         // Pin del LED interno que indica si hay conexión MQTT

// ===== CREACIÓN DE OBJETOS =====
WiFiClient espClient;               // Cliente WiFi
PubSubClient client(espClient);     // Cliente MQTT usando la conexión WiFi

// Variables para medir el tiempo entre envíos
unsigned long lastSendTime = 0;
unsigned long lastStatusTime = 0;
const unsigned long statusInterval = 5000;  // Intervalo de estado (5 segundos)

// ===== FUNCIÓN PARA CONECTARSE AL WiFi =====
void connectWiFi() {
  WiFi.disconnect(true);                  // Desconecta cualquier red previa
  WiFi.mode(WIFI_STA);                    // Modo estación (cliente)
  WiFi.config(local_ip, gateway, subnet, primaryDNS, secondaryDNS); // Configura IP fija
  WiFi.begin(ssid, password);             // Inicia conexión WiFi

  Serial.print("Conectando a WiFi");
  while (WiFi.status() != WL_CONNECTED) {  // Espera hasta que se conecte
    delay(500);
    Serial.print(".");
  }

  Serial.println("\nConectado! IP: " + WiFi.localIP().toString());
}

```

```
// ===== FUNCIÓN PARA CONECTARSE AL SERVIDOR MQTT =====
void connectMQTT() {
  while (!client.connected()) {                    // Mientras no esté conectado al broker
    Serial.print("Conectando a MQTT...");
    if (client.connect(clientID)) {                // Intenta conectarse con el ID
      Serial.println("Conectado!");
      digitalWrite(ledPin, HIGH);                  // Enciende el LED como indicador
    } else {
      Serial.print("Fallo, rc=");
      Serial.print(client.state());                // Muestra el código de error
      Serial.println(" Reintentando en 5 segundos...");
      delay(5000);                                 // Espera antes de intentar de nuevo
    }
  }
}

// ===== FUNCIÓN PARA OBTENER EL VOLTAJE PROMEDIO =====
float getAverageVoltage(int samples) {
  float sum = 0;
  for (int i = 0; i < samples; i++) {
    sum += analogRead(sensorPin) * vRef / adcMax;  // Convierte lectura ADC a voltaje
    delay(20);                                     // Pequeña espera entre lecturas
  }
  return sum / samples;                            // Retorna el promedio de voltajes
}

// ===== FUNCIÓN SETUP: se ejecuta una vez al encender el ESP32 =====
void setup() {
  Serial.begin(115200);               // Inicia la comunicación por puerto serie
  pinMode(ledPin, OUTPUT);            // Define el pin del LED como salida
  digitalWrite(ledPin, LOW);          // Apaga el LED al inicio
  analogReadResolution(12);           // Define resolución ADC a 12 bits

  connectWiFi();                      // Conecta al WiFi
  client.setServer(mqtt_server, mqtt_port); // Configura el servidor MQTT
  connectMQTT();                      // Conecta al broker MQTT
}

// ===== FUNCIÓN LOOP: se repite constantemente =====
void loop() {
  if (WiFi.status() != WL_CONNECTED) {
    connectWiFi();    // Si se pierde la conexión WiFi, intenta reconectar
  }

  if (!client.connected()) {
    digitalWrite(ledPin, LOW); // Apaga el LED si se desconecta de MQTT
    connectMQTT();             // Reintenta conectar al servidor MQTT
  }

  client.loop(); // Mantiene viva la conexión MQTT

  unsigned long currentMillis = millis(); // Tiempo actual desde que se encendió el ESP

```

```
  // Si pasaron 7 segundos desde el último envío de datos
  if (currentMillis - lastSendTime >= reportInterval) {
    lastSendTime = currentMillis;

    float voltage = getAverageVoltage(5);  // Lee el voltaje promedio del sensor
    float windKmh = voltage * 50.0;        // Convierte voltaje a velocidad del viento (aproximación)

    char payload[10];                      // Arreglo donde se guardará el dato en texto
    dtostrf(windKmh, 4, 1, payload);       // Convierte el número a texto con 1 decimal

    if (client.publish(mqtt_topic, payload)) { // Envía el dato por MQTT
      Serial.print("Velocidad enviada: ");
      Serial.print(payload);
      Serial.println(" km/h");
    }
  }

  // Si pasaron 5 segundos, se publica el estado "ON"
  if (currentMillis - lastStatusTime >= statusInterval) {
    lastStatusTime = currentMillis;
    client.publish(status_topic, "ON", true); // Publica estado de conexión
  }

  delay(100); // Pequeña pausa para no saturar el loop
}
```

Anexo 13. Código para la configuración del nodo sensor temperatura

```
#include <WiFi.h>                // Librería para conectarse a una red WiFi
#include <PubSubClient.h>        // Librería para comunicarse con un broker MQTT
#include <OneWire.h>             // Librería para comunicación 1-Wire (sensor DS18B20)
#include <DallasTemperature.h>   // Librería específica para sensores DS18B20

// ===== CONFIGURACIÓN DE RED WiFi =====
const char* ssid = "rolando 2.4G";          // Nombre de tu red WiFi
const char* password = "13121989";          // Contraseña del WiFi

// Dirección IP fija y parámetros de red
IPAddress local_ip(192, 168, 1, 23);        // IP que tendrá el ESP32
IPAddress gateway(192, 168, 1, 1);          // Dirección del router
IPAddress subnet(255, 255, 255, 0);         // Máscara de subred
IPAddress primaryDNS(8, 8, 8, 8);           // DNS primario (Google)
IPAddress secondaryDNS(8, 8, 4, 4);         // DNS secundario

// ===== CONFIGURACIÓN DE MQTT =====
const char* mqtt_server = "192.168.1.21";     // IP del servidor MQTT (ej: Mosquitto)
const int mqtt_port = 1883;                   // Puerto por defecto de MQTT
const char* mqtt_topic = "TEMPERATURA";       // Tema donde se enviará la temperatura
const char* status_topic = "estado/TEMPERATURA"; // Tema donde se enviará el estado del dispositivo
const char* clientID = "TEMPERATURA_ESP32";   // Nombre único del cliente MQTT

// ===== CONFIGURACIÓN DEL SENSOR DE TEMPERATURA (DS18B20) =====
#define ONE_WIRE_BUS 4              // Pin al que está conectado el sensor DS18B20
OneWire oneWire(ONE_WIRE_BUS);      // Crea una instancia de OneWire
DallasTemperature sensors(&oneWire); // Usa la librería Dallas con el bus OneWire

// ===== LED PARA INDICAR CONEXIÓN MQTT =====
const int ledPin = 2;  // Pin del LED (usualmente el LED azul del ESP32)

WiFiClient espClient;            // Cliente WiFi
PubSubClient client(espClient);  // Cliente MQTT usando la conexión WiFi

// Variables para controlar el tiempo entre envíos
unsigned long lastSendTime = 0;
const unsigned long sendInterval = 2000; // Enviar temperatura cada 2 segundos

unsigned long lastHeartbeatTime = 0;
const unsigned long heartbeatInterval = 5000; // Enviar estado "ON" cada 5 segundos

// ===== FUNCIÓN PARA CONECTARSE A LA RED WiFi =====
void connectWiFi() {
  WiFi.disconnect(true);                      // Borra conexiones anteriores
  WiFi.mode(WIFI_STA);                        // Modo estación (cliente)
  WiFi.config(local_ip, gateway, subnet, primaryDNS, secondaryDNS); // Configura IP fija
  WiFi.begin(ssid, password);                 // Intenta conectarse

  Serial.print("Conectando a WiFi");
  while (WiFi.status() != WL_CONNECTED) {     // Espera hasta que se conecte
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConectado! IP: " + WiFi.localIP().toString());
}
```

```
// ===== FUNCIÓN PARA CONECTARSE AL SERVIDOR MQTT =====
void connectMQTT() {
  while (!client.connected()) {                // Mientras no esté conectado
    Serial.print("Conectando a MQTT...");
    if (client.connect(clientID)) {            // Intenta conectar con el ID asignado
      Serial.println("Conectado!");
      digitalWrite(ledPin, HIGH);              // Enciende el LED si se conecta exitosamente
    } else {
      Serial.print("Fallo, rc=");
      Serial.print(client.state());            // Muestra código de error
      Serial.println(" Reintentando en 5 segundos...");
      delay(5000);                             // Espera antes de reintentar
    }
  }
}

// ===== CONFIGURACIÓN INICIAL (se ejecuta una sola vez al encender) =====
void setup() {
  Serial.begin(115200);          // Inicia la comunicación serie para monitoreo
  pinMode(ledPin, OUTPUT);       // Configura el pin del LED como salida
  digitalWrite(ledPin, LOW);     // Apaga el LED al inicio

  sensors.begin();               // Inicia el sensor de temperatura
  connectWiFi();                 // Se conecta al WiFi
  client.setServer(mqtt_server, mqtt_port); // Configura el servidor MQTT
  connectMQTT();                 // Se conecta al broker MQTT
}

// ===== BUCLE PRINCIPAL (se repite constantemente) =====
void loop() {
  if (WiFi.status() != WL_CONNECTED) {
    connectWiFi();               // Si se desconecta del WiFi, vuelve a conectarse
  }

  if (!client.connected()) {
    digitalWrite(ledPin, LOW);   // Apaga el LED si pierde la conexión MQTT
    connectMQTT();               // Reintenta la conexión MQTT
  }

  client.loop();                 // Mantiene viva la conexión con el broker

  unsigned long currentMillis = millis(); // Tiempo desde que se encendió el ESP32

  // ===== ENVIAR TEMPERATURA CADA 2 SEGUNDOS =====
  if (currentMillis - lastSendTime >= sendInterval) {
    lastSendTime = currentMillis;

    sensors.requestTemperatures();         // Pide la lectura al sensor
    float tempC = sensors.getTempCByIndex(0); // Obtiene temperatura del primer sensor

    char payload[10];
    dtostrf(tempC, 4, 2, payload);         // Convierte el número en texto con 2 decimales

    if (client.publish(mqtt_topic, payload)) { // Envía la temperatura al broker
      Serial.print("Temperatura enviada: ");
      Serial.print(payload);
      Serial.println(" °C");
    } else {
      Serial.println("❌ Error al publicar temperatura");
    }
  }

  // ===== ENVIAR ESTADO "ON" CADA 5 SEGUNDOS =====
  if (currentMillis - lastHeartbeatTime >= heartbeatInterval) {
    lastHeartbeatTime = currentMillis;
    client.publish(status_topic, "ON", true); // Publica mensaje indicando que el dispositivo está activo
  }
}
```

Anexo 14. Código para la configuración del nodo sensor temperatura infrarrojo

```
#include <Wire.h>                    // Librería para comunicación I2C
#include <WiFi.h>                    // Librería para conectar a una red WiFi
#include <Adafruit_MLX90614.h>      // Librería para el sensor de temperatura infrarrojo MLX90614
#include <PubSubClient.h>           // Librería para conectarse a un broker MQTT

// ===== CONFIGURACIÓN DE LA RED WiFi =====
const char* ssid = "rolando 2.4G";        // Nombre del WiFi
const char* password = "13121989";        // Contraseña del WiFi

// ===== CONFIGURACIÓN DE RED ESTÁTICA =====
IPAddress local_IP(192, 168, 1, 99);      // IP fija del ESP32
IPAddress gateway(192, 168, 1, 1);        // Puerta de enlace (router)
IPAddress subnet(255, 255, 255, 0);       // Máscara de subred
IPAddress primaryDNS(8, 8, 8, 8);         // DNS primario (Google)
IPAddress secondaryDNS(8, 8, 4, 4);       // DNS secundario

// ===== CONFIGURACIÓN MQTT =====
const char* mqtt_server = "192.168.1.21";       // IP del broker MQTT
const int mqtt_port = 1883;                     // Puerto MQTT (por defecto)
const char* mqtt_topic = "PIR";                 // Tema donde se publicará la temperatura
const char* status_topic = "estado/PIR";        // Tema donde se enviará el estado del dispositivo
const char* clientID = "PIR_ESP32";             // ID del cliente MQTT

WiFiClient espClient;              // Cliente WiFi
PubSubClient client(espClient);    // Cliente MQTT usando el cliente WiFi

Adafruit_MLX90614 mlx = Adafruit_MLX90614(); // Crea un objeto para el sensor MLX90614

// ===== LED DE ESTADO =====
const int ledPin = 2;  // Pin del LED interno del ESP32 (útil para indicar conexión)

// ===== TEMPORIZADORES =====
unsigned long lastTempTime = 0;             // Último envío de temperatura
unsigned long lastStatusTime = 0;           // Último envío de estado
const unsigned long tempInterval = 10000;   // Cada 10 segundos se envía temperatura
const unsigned long statusInterval = 5000;  // Cada 5 segundos se envía "ON" al estado

  connectWiFi();                    // Conecta a WiFi
  client.setServer(mqtt_server, mqtt_port); // Configura el broker MQTT
  connectMQTT();                    // Se conecta al broker MQTT
}

// ===== BUCLE PRINCIPAL =====
void loop() {
  if (WiFi.status() != WL_CONNECTED) {
    connectWiFi();    // Reconecta al WiFi si se perdió la conexión
  }

  if (!client.connected()) {
    digitalWrite(ledPin, LOW); // Apaga LED si no hay conexión MQTT
    connectMQTT();             // Reconecta al broker MQTT
  }

  client.loop();               // Mantiene viva la conexión MQTT

  unsigned long now = millis(); // Tiempo actual desde que encendió
```

```
68    // ===== ENVIAR TEMPERATURA CADA 10 SEGUNDOS =====
69    if (now - lastTempTime >= tempInterval) {
70      lastTempTime = now;
71
72      double temp_obj = mlx.readObjectTempC();  // Lee la temperatura del objeto en °C
73      char tempString[10];
74      dtostrf(temp_obj, 4, 2, tempString);      // Convierte la temperatura a texto con 2 decimales
75
76      if (client.publish(mqtt_topic, tempString)) {
77        Serial.print("Temperatura enviada: ");
78        Serial.print(tempString);
79        Serial.println(" °C");
80      }
81    }
82
83    // ===== ENVIAR ESTADO CADA 5 SEGUNDOS =====
84    if (now - lastStatusTime >= statusInterval) {
85      lastStatusTime = now;
86      client.publish(status_topic, "ON", true);  // Publica mensaje de "sigo activo"
87    }
88
89    delay(100); // Pausa corta para no saturar el microcontrolador
90  }
91
92  // ===== CONECTAR AL WiFi =====
93  void connectWiFi() {
94    WiFi.disconnect(true); // Elimina cualquier configuración anterior
95    WiFi.mode(WIFI_STA);   // Modo estación (cliente)
96
97    // Intenta configurar IP fija
98    if (!WiFi.config(local_IP, gateway, subnet, primaryDNS, secondaryDNS)) {
99      Serial.println("Error al configurar IP estática");
100   }
101
102   WiFi.begin(ssid, password); // Conecta al WiFi
103   Serial.print("Conectando a WiFi");
104
105   while (WiFi.status() != WL_CONNECTED) {
106     delay(500);
107     Serial.print(".");
108   }
109
110   Serial.println("\nConectado! IP: " + WiFi.localIP().toString());
111 }
112
113 // ===== CONECTAR AL BROKER MQTT =====
114 void connectMQTT() {
115   while (!client.connected()) {
116     Serial.print("Conectando a MQTT...");
117
118     if (client.connect(clientID)) {            // Conecta al broker con el ID asignado
119       Serial.println("Conectado!");
120       client.publish(status_topic, "ON", true); // Publica estado inicial
121       digitalWrite(ledPin, HIGH);               // Enciende el LED como señal de conexión
122     } else {
123       Serial.print("Fallo, rc=");
124       Serial.print(client.state());             // Imprime código de error
125       Serial.println(" Reintentando en 5 segundos...");
126       delay(5000);                              // Espera antes de reintentar
127     }
128   }
129 }
```

Anexo 15. Código para la configuración del nodo sensor humedad de suelo V1.2

```
#include <WiFi.h>           // Librería para conectarse al WiFi
#include <PubSubClient.h>   // Librería para conectarse al servidor MQTT

// ===== PINES =====
const int sensorPin = 32;   // Pin donde está conectado el sensor de humedad de suelo
const int ledPin = 2;       // Pin del LED (usado como indicador de conexión)

// ===== CONFIGURACIÓN WiFi =====
const char* ssid = "rolando 2.4G";     // Nombre de tu red WiFi
const char* password = "13121989";     // Contraseña de tu red WiFi

// ===== IP ESTÁTICA =====
IPAddress local_ip(192, 168, 1, 19);     // IP fija que usará este ESP32
IPAddress gateway(192, 168, 1, 1);       // Puerta de enlace (router)
IPAddress subnet(255, 255, 255, 0);      // Máscara de subred
IPAddress primaryDNS(8, 8, 8, 8);        // DNS primario (Google)
IPAddress secondaryDNS(8, 8, 4, 4);      // DNS secundario

// ===== CONFIGURACIÓN MQTT =====
const char* mqtt_server = "192.168.1.21";         // IP de tu broker MQTT
const int mqtt_port = 1883;                       // Puerto MQTT
const char* mqtt_topic = "SENSOR_HUMEDAD";        // Tema donde se enviará el valor de humedad
const char* status_topic = "estado/HUMEDAD";      // Tema donde se enviará el estado del dispositivo
const char* clientID = "HUMEDAD_ESP32";           // Nombre único del cliente MQTT

// ===== CLIENTES WiFi y MQTT =====
WiFiClient espClient;              // Cliente para conexión WiFi
PubSubClient client(espClient);    // Cliente MQTT basado en la conexión WiFi

// ===== TEMPORIZADORES =====
const int reportInterval = 8000;     // Intervalo para enviar humedad (8 segundos)
unsigned long lastSendTime = 0;      // Última vez que se envió humedad
unsigned long lastStatusTime = 0;    // Última vez que se envió "ON"
const unsigned long statusInterval = 5000; // Enviar "ON" cada 5 segundos

// ===== SETUP (Se ejecuta una vez al encender el ESP32) =====
void setup() {
  Serial.begin(115200);             // Inicia la comunicación por el monitor serie
  pinMode(ledPin, OUTPUT);          // Define el LED como salida
  analogReadResolution(12);         // Usa resolución de 12 bits (0-4095)

  connectWiFi();                    // Conecta al WiFi
  client.setServer(mqtt_server, mqtt_port); // Define el broker MQTT
  connectMQTT();                    // Conecta al servidor MQTT
}

```

```
// ===== LOOP (Se ejecuta continuamente) =====
void loop() {
  if (WiFi.status() != WL_CONNECTED) {
    connectWiFi();   // Reconecta si se pierde conexión WiFi
  }

  if (!client.connected()) {
    digitalWrite(ledPin, LOW);  // Apaga el LED si no está conectado al broker
    connectMQTT();              // Intenta reconectar al broker MQTT
  }

  client.loop();  // Mantiene viva la conexión MQTT

  unsigned long now = millis(); // Tiempo actual desde que encendió el ESP32

  // ===== PUBLICAR HUMEDAD DEL SUELO CADA 8 SEGUNDOS =====
  if (now - lastSendTime >= reportInterval) {
    lastSendTime = now;

    int humedad = leerHumedadPrecisa();   // Lee y calcula humedad con precisión

    char payload[10];
    snprintf(payload, sizeof(payload), "%d", humedad); // Convierte el número a texto

    if (client.publish(mqtt_topic, payload)) {
      Serial.print("Humedad del suelo enviada: ");
      Serial.print(payload);
      Serial.println(" %");
    }
  }

  // ===== PUBLICAR ESTADO "ON" CADA 5 SEGUNDOS =====
  if (now - lastStatusTime >= statusInterval) {
    lastStatusTime = now;
    client.publish(status_topic, "ON", true);  // Informa que el dispositivo sigue activo
  }

  delay(100); // Pequeña pausa para evitar sobrecarga
}

// ===== CONECTAR AL WiFi =====
void connectWiFi() {
  WiFi.disconnect(true);                  // Borra conexiones anteriores
  WiFi.mode(WIFI_STA);                    // Modo estación
  WiFi.config(local_ip, gateway, subnet, primaryDNS, secondaryDNS); // IP fija
  WiFi.begin(ssid, password);             // Inicia conexión con el WiFi

   Serial.print("Conectando a WiFi");
   while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
   }
   Serial.println("\nConectado! IP: " + WiFi.localIP().toString());
}
```

```
// ===== CONECTAR AL BROKER MQTT =====
void connectMQTT() {
  while (!client.connected()) {
    Serial.print("Conectando a MQTT...");

    if (client.connect(clientID)) {
      Serial.println("Conectado!");
      client.publish(status_topic, "ON", true);  // Publica estado inicial
      digitalWrite(ledPin, HIGH);                // Enciende el LED al conectar
    } else {
      Serial.print("Fallo, rc=");
      Serial.print(client.state());              // Código de error MQTT
      Serial.println(" Reintentando en 5 segundos...");
      delay(5000);
    }
  }
}

// ===== LECTURA PRECISA DEL SENSOR DE HUMEDAD =====
int leerHumedadPrecisa() {
  const int muestras = 100;     // Cantidad de lecturas para promediar
  long suma = 0;

  for (int i = 0; i < muestras; i++) {
    suma += analogRead(sensorPin); // Lectura analógica
    delay(2);                      // Pausa pequeña entre lecturas
  }

  int promedio = suma / muestras; // Promedio de las lecturas

  // ===== Calibración del sensor =====
  // 1300 = suelo muy húmedo, 2500 = seco (ajustable)
  int humedad = map(promedio, 1300, 2500, 100, 0); // Convierte a % de humedad
  humedad = constrain(humedad, 0, 100);            // Limita entre 0 y 100 %

  return humedad;
}
```

Anexo 16. Código para la configuración del nodo sensor proximidad

```
#include <WiFi.h>            // Librería para conectarse a redes WiFi
#include <PubSubClient.h>    // Librería para conectarse a un broker MQTT

// ===== CONFIGURACIÓN DE PINES =====
#define TRIG_PIN 5   // Pin de disparo (TRIG) del sensor ultrasónico HC-SR04
#define ECHO_PIN 18  // Pin de eco (ECHO) del sensor ultrasónico
#define LED_PIN 2    // Pin del LED de estado (usualmente el LED interno del ESP32)

// ===== DATOS DE LA RED WiFi =====
const char* ssid = "rolando 2.4G";       // Nombre de tu red WiFi
const char* password = "13121989";       // Contraseña de tu red WiFi

// ===== CONFIGURACIÓN DE RED ESTÁTICA =====
IPAddress local_ip(192, 168, 1, 24);     // IP fija para el ESP32
IPAddress gateway(192, 168, 1, 1);       // Puerta de enlace (generalmente el router)
IPAddress subnet(255, 255, 255, 0);      // Máscara de subred
IPAddress primaryDNS(8, 8, 8, 8);        // DNS primario (Google)
IPAddress secondaryDNS(8, 8, 4, 4);      // DNS secundario

// ===== DATOS DEL BROKER MQTT =====
const char* mqtt_server = "192.168.1.21";          // IP del broker MQTT (local)
const int mqtt_port = 1883;                        // Puerto estándar de MQTT
const char* mqtt_topic = "PROXIMIDAD";             // Tema donde se publica la distancia
const char* status_topic = "estado/PROXIMIDAD";    // Tema donde se publica el estado ("ON")
const char* clientID = "PROXIMIDAD_ESP32";         // Identificador único del cliente MQTT

WiFiClient espClient;              // Cliente WiFi
PubSubClient client(espClient);    // Cliente MQTT que usa WiFi

// ===== TEMPORIZADORES =====
unsigned long lastDistanceTime = 0;
const unsigned long distanceInterval = 15000;   // Medir y enviar distancia cada 15 segundos

unsigned long lastStatusTime = 0;
const unsigned long statusInterval = 5000;      // Enviar "ON" cada 5 segundos

// ===== CONEXIÓN A WiFi =====
void connectWiFi() {
  WiFi.disconnect(true);                     // Borra configuraciones previas
  WiFi.mode(WIFI_STA);                       // Modo estación (cliente WiFi)
  WiFi.config(local_ip, gateway, subnet, primaryDNS, secondaryDNS); // IP fija
  WiFi.begin(ssid, password);                // Conexión a WiFi

  Serial.print("Conectando a WiFi");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nConectado a WiFi. IP: " + WiFi.localIP().toString());
}

// ===== CONEXIÓN AL BROKER MQTT =====
void connectMQTT() {
  while (!client.connected()) {
    Serial.print("Conectando a MQTT...");

    if (client.connect(clientID)) {
      Serial.println("Conectado!");
      digitalWrite(LED_PIN, HIGH);           // Enciende el LED si la conexión fue exitosa
```

```
    } else {
      Serial.print("Fallo, rc=");
      Serial.print(client.state());          // Muestra código de error si falla
      Serial.println(" Reintentando en 5 segundos...");
      delay(5000);
    }
  }
}

// ===== FUNCIÓN PARA MEDIR DISTANCIA =====
float measureDistance() {
  digitalWrite(TRIG_PIN, LOW);        // Asegura que TRIG esté en bajo
  delayMicroseconds(2);               // Espera corta
  digitalWrite(TRIG_PIN, HIGH);       // Envía un pulso de 10µs a TRIG
  delayMicroseconds(10);
  digitalWrite(TRIG_PIN, LOW);

  // Lee el tiempo que tarda en volver el eco. Si tarda más de 30ms, devuelve -1
  long duration = pulseIn(ECHO_PIN, HIGH, 30000);
  if (duration == 0) return -1;

  // Calcula la distancia en centímetros (sonido viaja a 0.034 cm/µs)
  return duration * 0.034 / 2;
}

// ===== SETUP (EJECUTADO UNA VEZ AL INICIO) =====
void setup() {
  Serial.begin(115200);            // Comunicación con el monitor serial
  pinMode(TRIG_PIN, OUTPUT);       // TRIG como salida
  pinMode(ECHO_PIN, INPUT);        // ECHO como entrada
  pinMode(LED_PIN, OUTPUT);        // LED como salida
  digitalWrite(LED_PIN, LOW);      // Apaga el LED al inicio

  connectWiFi();                   // Conexión WiFi
  client.setServer(mqtt_server, mqtt_port);  // Configura el broker MQTT
  connectMQTT();                   // Conexión inicial al broker
}

// ===== LOOP PRINCIPAL =====
void loop() {
  if (WiFi.status() != WL_CONNECTED) {
    connectWiFi();   // Reintenta conexión WiFi si se pierde
  }

  if (!client.connected()) {
    digitalWrite(LED_PIN, LOW);  // Apaga el LED si pierde conexión MQTT
    connectMQTT();               // Reintenta conexión al broker
  }

  client.loop();  // Mantiene activa la conexión MQTT

  unsigned long currentMillis = millis(); // Tiempo actual desde el arranque

```

```
  // ===== ENVIAR DISTANCIA CADA 15 SEGUNDOS =====
  if (currentMillis - lastDistanceTime >= distanceInterval) {
    lastDistanceTime = currentMillis;

    float distance = measureDistance();   // Mide la distancia
    char payload[10];
    dtostrf(distance, 4, 1, payload);     // Convierte la distancia a texto con 1 decimal

    if (client.publish(mqtt_topic, payload)) {
      Serial.print("Distancia publicada: ");
      Serial.print(payload);
      Serial.println(" cm");
    } else {
      Serial.println("❌ Error al publicar la distancia");
    }
  }

  // ===== ENVIAR "ON" CADA 5 SEGUNDOS =====
  if (currentMillis - lastStatusTime >= statusInterval) {
    lastStatusTime = currentMillis;
    client.publish(status_topic, "ON", true); // Señal de que el ESP32 sigue activo
  }
}
```

Anexo 17. Código para la configuración del nodo sensor bomba de agua

```
#include <WiFi.h>
#include <PubSubClient.h>

// Credenciales WiFi
const char* ssid = "rolando 2.4G";
const char* password = "13121989";

// Configuración del servidor MQTT
const char* mqtt_server = "192.168.235.21";
const int mqtt_port = 1883;
const char* mqtt_topic = "VALVULA";
const char* status_topic = "estado/VALVULA";
const char* clientID = "VALVULA_ESP32";

// Pin de la válvula (relé)
#define VALVE_PIN 27

WiFiClient espClient;
PubSubClient client(espClient);

// Configuración de IP estática
IPAddress local_IP(192, 168, 235, 98);
IPAddress gateway(192, 168, 235, 108);
IPAddress subnet(255, 255, 255, 0);
IPAddress primaryDNS(8, 8, 8, 8);
IPAddress secondaryDNS(8, 8, 4, 4);

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

  // Configurar pin como salida
  pinMode(VALVE_PIN, OUTPUT);
  digitalWrite(VALVE_PIN, HIGH); // Relé desactivado al iniciar (activo en bajo)

  connectWiFi();

  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);

  connectMQTT();
}

void loop() {
  if (!client.connected()) {
    connectMQTT();
  }
  client.loop();
}

void connectWiFi() {
  // Configura IP estática
  if (!WiFi.config(local_IP, gateway, subnet, primaryDNS, secondaryDNS)) {
    Serial.println("Fallo al configurar IP estática");
  }

  WiFi.begin(ssid, password);
  Serial.print("Conectando a WiFi");

  while (WiFi.status() != WL_CONNECTED) {
```

```
    delay(500);
    Serial.print(".");
  }

  Serial.println("\nConectado a WiFi con IP fija: " + WiFi.localIP().toString());
}

void connectMQTT() {
  while (!client.connected()) {
    if (client.connect(clientID, nullptr, nullptr, status_topic, 0, true, "OFF")) {
      Serial.println("Conectado a MQTT");
      client.subscribe(mqtt_topic);
      client.publish(status_topic, "OFF", true); // Publicar estado inicial
    } else {
      Serial.println("Fallo al conectar al broker MQTT. Reintentando...");
      delay(5000);
    }
  }
}

void callback(char* topic, byte* payload, unsigned int length) {
  String message;
  for (unsigned int i = 0; i < length; i++) {
    message += (char)payload[i];
  }
  message.trim();
  message.toUpperCase();

  if (message == "ON") {
    digitalWrite(VALVE_PIN, LOW);  // Activar relé (activo en bajo)
    client.publish(status_topic, "ON", true);
    Serial.println("Válvula activada");
  } else if (message == "OFF") {
    digitalWrite(VALVE_PIN, HIGH); // Desactivar relé
    client.publish(status_topic, "OFF", true);
    Serial.println("Válvula desactivada");
  }
}
```