Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Desarrollo de un nodo de sensado de bajo costo para la recopilación de datos ambientales

Informe de Trabajo Final de Graduación para optar por el titulo de Ingeniero en Electrónica con el grado académico de Licenciatura

Presenta:

Federico A. Rivera Moya

Cartado, Costa Rica

26 de noviembre de 2025

INSTITUTO TECNOLÓGICO DE COSTA RICA

ESCUELA DE INGENIERÍA ELECTRÓNICA

TRABAJO FINAL DE GRADUACIÓN

ACTA DE APROBACIÓN

Defensa del Trabajo Final de Graduación
Requisito para optar por el título de Ingeniero en Electrónica
Grado Académico de Licenciatura
Instituto Tecnológico de Costa Rica

El Tribunal Evaluador aprueba la defensa del Trabajo Final de Graduación denominado Desarrollo de un nodo de sensado de bajo costo para la recopilación de datos ambientales, realizado por el señor Federico Andrés Rivera Moya, y hace constar que cumple con las normas establecidas por la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal Evaluador

SERGIO ARTURO MORALES HERNANDEZ (FIRMA)
PERSONA FISICA, CPF-02-0455-0934.
Fecha declarada: 01/12/2025 06:46:20 PM
Esta es una representación gráfica únicamente, verifique la validez de la firma.

Ing. Sergio Morales Hernández

Profesor lector

TEC
Firmado digitalmente por HUGO ANDRES SANCHEZ ORTIZ (FIRMA)
Fecha: 2025.12.01 12:56:52 +01'00'

Ing. Hugo Sánchez Ortiz

Profesor lector

PAOLA VEGA CASTILLO (FIRMA)
PERSONA FISICA, CPF-01-0937-0493.
Fecha declarada: 02/12/2025 10:56:06 AM
Esta representación visual no es fuente de confianza. Valide siempre la firma.

Dr.-Ing. Paola Vega Castillo

Profesor asesor

Cartago, 26 de noviembre de 2025

INSTITUTO TECNOLÓGICO DE COSTA RICA
ESCUELA DE INGENIERÍA ELECTRÓNICA
TRABAJO FINAL DE GRADUACIÓN
TRIBUNAL EVALUADOR
ACTA DE EVALUACIÓN

Defensa del Trabajo Final de Graduación
Requisito para optar por el título de Ingeniero en Electrónica
Grado Académico de Licenciatura
Instituto Tecnológico de Costa Rica

Estudiante: Federico Andrés Rivera Moya Carné: 2013105755

Nombre del Trabajo Final de Graduación: *Desarrollo de un nodo de sensado de bajo costo para la recopilación de datos ambientales.*

Los miembros de este Tribunal hacen constar que este Trabajo Final de Graduación ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica

Nota de Trabajo Final de Graduación: 100

Miembros del Tribunal

SERGIO ARTURO MORALES HERNANDEZ (FIRMA)
PERSONA FISICA, CPF-02-0455-0934.
Fecha declarada: 01/12/2025 06:45:28 PM
Esta es una representación gráfica únicamente, verifique la validez de la firma.

Ing. Sergio Morales Hernández

Profesor lector

TEC
Firmado digitalmente por HUGO ANDRES SANCHEZ ORTIZ (FIRMA)
Fecha: 2025.12.01 12:57:49 +01'00'

Ing. Hugo Sánchez Ortiz

Profesor lector

PAOLA VEGA CASTILLO (FIRMA)
PERSONA FISICA, CPF-01-0937-0493.
Fecha declarada: 02/12/2025 10:56:42 AM
Esta representación visual no es fuente de confianza. Valide siempre la firma.

Dr.-Ing. Paola Vega Castillo

Profesor asesor

Cartago, 26 de noviembre, 2025

1. Declaratoria de autenticidad

Yo, Federico Andrés Rivera Moya, carnet 2013105755, estudiante de la carrera de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica, declaro bajo juramento que el presente documento titulado "Desarrollo de un nodo de sensado de bajo costo para la recopilación de datos ambientales", es el resultado de mi trabajo personal.

Manifiesto que las ideas, análisis, resultados, conclusiones y propuestas contenidas en este documento son de mi autoría, basándome en herramientas y conocimientos adquiridos en la carrera, así como en fuentes de información debidamente citadas y referenciadas de acuerdo con las normas académicas establecidas.

Asimismo, afirmo que no he incurrido en plagio, copia no autorizada o uso indebido de información, y que asumo la total responsabilidad por el contenido de este trabajo.

2. Agradecimientos

A mi familia, por su apoyo incondicional, su paciencia y por acompañarme en cada paso de esta etapa. Su confianza y motivación constante fueron fundamentales para alcanzar esta meta.

A mi novia, por su comprensión, cariño y por estar presente incluso en los momentos más exigentes del proceso. Su apoyo y ánimo fueron una fuente constante de inspiración y motivación.

A mi asesora, la Dra.-Ing. Paola Vega Castillo, por su orientación, dedicación y valiosos consejos para lograr el desarrollo de este proyecto.

A mis compañeros y amigos de carrera, por su colaboración, las ideas compartidas y los momentos de esfuerzo conjunto que enriquecieron esta experiencia.

Finalmente, al Instituto Tecnológico de Costa Rica y a la Escuela de Ingeniería Electrónica, por la formación brindada y por promover espacios que impulsan el desarrollo de proyectos con impacto en la sociedad.

3. Resumen

Este trabajo presenta el desarrollo de un sistema de sensado ambiental de bajo costo para medir parámetros de calidad de aire, temperatura, humedad y ruido en entornos urbanos. Surge como respuesta a la necesidad de sistemas accesibles y replicables que permitan obtener información ambiental con mayor cobertura que las opciones actuales, proveyendo datos actualizados con un potencial de aplicación en el contexto de Costa Rica como destino turístico inteligente. El sistema integra sensores comerciales controlados por un microcontrolador basado en ESP32S3, que transmite los datos recolectados a una central mediante comunicación LoRa y los publica en la nube a través de la plataforma Adafruit IO. Se desarrolló el diseño electrónico del sistema, la programación modular del mismo y pruebas de campo y autonomía para evaluar estabilidad y fiabilidad de la comunicación, además del posible tiempo de actividad del sistema. Los resultados evidencian una transmisión estable con RSSI entre –70 y –50 dBm y SNR entre 9 y 12 dB, con concordancia entre los datos recibidos y los registrados en la nube. El sistema demuestra ser una alternativa de accesible costo, escalable y eficiente, estableciendo una base para la futura implementación de una red de monitoreo ambiental.

Palabras Clave: ESP32S3, TVCO, eCO_2, AQI, material particulado, LoRa, SNR, RSSI, Adafruit IO

4. Abstract

This work presents the development of a low-cost environmental sensing system designed to measure air quality, temperature, humidity, and noise parameters in urban environments. It arises in response to the need of accessible and replicable systems capable of providing environmental information with broader coverage than current alternatives, offering updated data with potential applications within the context of Costa Rica as a smart tourist destination. The system integrates commercial sensors controlled by an ESP32S3-based microcontroller, which transmits the collected data to a central unit via LoRa communication and publishes them on the cloud through the Adafruit IO platform. The project involved the electronic design of the system, its modular programming, and field and autonomy tests to evaluate communication stability, reliability, and operational lifetime. The results show stable transmission with RSSI values between –70 and –50 dBm and SNR values between 9 and 12 dB, with strong consistency between the received and cloud-stored data. The system proves to be a cost-effective, scalable, and efficient alternative, establishing a baseline for the future creation of an environmental monitoring network.

Keywords: ESP32, TVCO, eCO_2, AQI, Particulate matter, LoRa, SNR, RSSI, Adafruit IO

Índice

Índice de figuras

Índice de cuadros

5. Introducción

5.1. Entorno del proyecto

Esta propuesta de proyecto es un resultado del proyecto Destinos Turísticos Inteligentes: Tecnologías de Información y Comunicación Aplicadas al Turismo, coordinado por el Instituto Tecnológico de Costa Rica y cofinanciado por el Fondo de Cooperación Triangular Unión Europea - Costa Rica - Latinoamérica y el Caribe [1]. En este proyecto se realizó un análisis sobre cuatro cantones piloto en Costa Rica para determinar qué brechas deben atender para convertirse en destinos turísticos inteligentes. Una de las tendencias observadas es la digitalización impulsada por la pandemia del COVID-19. Esta ha cambiado la actitud de la sociedad hacia el turismo y la información climatológica; la población se ha caracterizado por estar más informada y pendiente de información ambiental. Esto implica la necesidad de poder obtener datos climatológicos y atmosféricos para que estos puedan estar a disposición de las entidades encargadas del manejo y control ambiental, como el Ministerio Nacional de Ambiente y Energía (MINAE) y sus subdivisiones, los gobiernos locales y también la población interesada en dicha información.

En el proyecto mencionado anteriormente se realizó el Diagnóstico y Recomendaciones para un Plan Estratégico de Turismo Inteligente en Costa Rica del 2024 [2]. Este estudio aborda íntegramente las dimensiones sociales, físicas, ambientales, económicas e institucionales de las ciudades y de los territorios de San Carlos, Sarchí, Tarrazú y Tibás. Este califica a estos cantones bajo una escala del 1 al 10, donde 1 es la peor puntuación e indica un nivel de madurez bajo en el área siendo estudiada. Dado el enfoque del presente Trabajo Final de Graduación, el dato más relevante sería el de ambiente, el cual recibe una calificación de 3.4 (madurez medio-bajo) en promedio para los cantones estudiados; específicamente se resaltan el factor aire, con una calificación de 1.88, y el factor ruido, con la misma calificación. Este estudio también determina que en estos cantones no existe un sistema capaz de proveer datos ambientales a las municipalidades, a la población o a los turistas, el cual es una de las condiciones que se debe cumplir para que se les pueda considerar como destino turístico inteligente y para poder mantener una mejor gestión del territorio. Como parte del mismo diagnóstico se diseñó una Hoja de Ruta [3], la cual indica que una de las acciones a realizar es la implementación de un sistema de sensado para la recopilación de datos ambientales como calidad del aire, temperatura, humedad y niveles de contaminación acústica.

Recientemente, el Ministerio de Salud, junto con el Instituto Meteorológico Nacional (IMN) y el MINAE, oficializaron el Convenio Marco de Cooperación para el Monitoreo de la Calidad del Aire en Costa Rica. Esto marca el primer paso de implementación de estrategias para este tipo de sistemas en el país, mediante la instalación de nodos de sensado de Índice de Calidad del Aire (AQI). Esta primera iteración se encuentra limitada a 7 puntos específicos, proveyendo baja cobertura y siendo potencialmente de alto costo en su implementación. Esto presenta una oportunidad para el desarrollo de nodos de bajo costo para implementar una red de sensado más general, de precio más accesible, de fácil replicación y escalabilidad, que posea la capacidad de medir las condiciones generales de calidad del aire, niveles de ruido, temperatura y humedad, con una cobertura mucho más amplia que la estrategia actual, para atender las necesidades de los destinos turísticos inteligentes. Es importante que el sistema que se visualiza sea parte de una amplia red capaz de comunicarse de manera inalámbrica con cada nodo de esta y con una central que maneje los datos recopilados para que estos puedan ser accedidos y visualizados por los actores públicos y la ciudadanía. Esto puede contribuir a mejorar la toma de decisiones, la gestión del territorio y de proyectos ambientales.

El Air Quality Index (AQI) es un indicador desarrollado y adoptado por gobiernos internacionales que evalúa la calidad del aire basado en los niveles de presencia de gases nocivos. Según el AQI, los principales contaminantes atmosféricos, en cuanto a gases y partículas, se refieren a ozono (O3), monóxido de carbono (CO), dióxido de azufre (SO2), dióxido de nitrógeno (NO_2) y contaminación por partículas de tamaño menor a 10 µm y 2.5 µm (PM10 y PM2.5, respectivamente). Esto también es indicado por el decreto Nº 30221-S, o *"Reglamento sobre Inmisión de Contaminantes Atmosféricos"* [4], y el decreto N° 43184-S-MINAE, o *"Reglamento sobre emisión de contaminantes atmosféricos provenientes de calderas y hornos de tipo directo e indirecto"* [6], los cuales indican los niveles aceptables para la emisión de estos contaminantes en Costa Rica. Además, el Diagnóstico y Recomendaciones para un Plan Estratégico de Turismo Inteligente del 2024 también indica altos niveles de contaminación acústica, por lo que este es un factor que también se debe tomar en cuenta. Se destaca también la necesidad de poder realizar mediciones de temperatura y humedad para un desglose de información más completo y relevante a necesidades turísticas.

Una manera moderna de integrar tantos componentes y cumplir con los requerimientos de comunicación es mediante el uso de un microcontrolador programable; existen varias opciones en el mercado, pero recientemente resaltan los sistemas ESP32. Estos poseen una amplia gama de interfaces de comunicación como circuitos inter-integrados (I2C), interfaz periférica serial (SPI), transmisor/receptor asíncrono universal (UART) y convertidor analógico-digital (ADC), haciéndolos capaces de manejar una amplia gama de entradas y salidas de diferentes dispositivos. Estos en muchos casos poseen también conectividad Wi-Fi y Bluetooth integradas, además de que existen variantes equipa-

das con sistemas de radiofrecuencia LoRa, permitiendo comunicaciones inalámbricas a larga distancia. Estas características hacen al ESP32 ideal para aplicaciones IoT, ya que permite la integración sencilla de múltiples sensores como los de temperatura, humedad, calidad del aire, entre otros, en un solo dispositivo, además presenta bajo consumo energético y buena capacidad de procesamiento.

Se resalta la integración de los sistemas RF LoRa en los módulos ESP32. LoRa es una tecnología de modulación de radiofrecuencia diseñada para comunicaciones de largo alcance. Se basa en un tipo especial de modulación de espectro ensanchado llamada Chirp Spread Spectrum (CSS), que permite transmitir datos a distancias largas con alta inmunidad al ruido y bajo consumo de energía. En Costa Rica estos sistemas operan en la banda de 433 MHz o 902-928 MHz; estas bandas son reservadas para uso de radioaficionados, haciendo la comunicación LoRa accesible y libre para su uso en el desarrollo de aplicaciones IoT.

Existen en el mercado diversos sensores de calidad del aire, ruido y temperatura compatibles con módulos ESP32, caracterizados por su amplia disponibilidad y precios accesibles, lo que los hace ideales para aplicaciones de bajo costo. Estos sensores varían en precio y precisión según el tipo y la tecnología empleada, pero por lo general son suficientes para su empleo en aplicaciones de monitoreo ambiental en ciudades inteligentes o entornos domésticos. La mayoría utiliza interfaces comunes como I2C, UART o analógica, lo que facilita su integración con el ESP32. En general, ofrecen un buen equilibrio entre costo, consumo energético y confiabilidad en mediciones básicas.

5.2. Definición del problema

5.2.1. Generalidades

Según el Diagnóstico y Recomendaciones para un Plan Estratégico de Turismo Inteligente en Costa Rica del 2024 [2] realizado para el proyecto Destinos Turísticos Inteligentes, en general, y a pesar de recientes desarrollos, Costa Rica aún posee una baja madurez en cuanto al desarrollo e implementación de estrategias efectivas para el control, protección y manejo sostenible del ambiente en el que se desenvuelve la población. La adopción de sistemas inteligentes de esta índole no solo podría llevar a un aumento en el tráfico turístico del país, sino que también impulsaría una tendencia de mejoramiento en la eficacia en el desarrollo sostenible de destinos turísticos y ciudades, conllevando, por ende, a un mejoramiento en la calidad de vida de los residentes del país y su transformación en un destino turístico inteligente.

Este mismo diagnóstico resalta la importancia de implementar sistemas de recopilación de datos y plataformas digitales que permitan la integración de información para facilitar la toma de decisiones. Para un destino turístico de alto perfil como Costa Rica,

es de gran importancia poder obtener información ambiental actualizada y de rápido acceso, con una amplia cobertura en la recopilación de estos datos. Dicho esto, existen varias trabas en la adopción de sistemas similares a los que están empezando a ser implementados en el país, como precios que pueden ser desde los altos cientos hasta los miles de dólares por nodo, disparidad de rango de cobertura versus costo, poca escalabilidad y difícil replicación de nodos adicionales.

Con base en lo anterior, se da paso a la propuesta de este proyecto: el desarrollo de un nodo de sensado de bajo costo, escalable y de fácil replicación para la futura implementación de una red con mayor cobertura que alternativas actuales. Este debe poder medir datos atmosféricos como AQI, temperatura, humedad y niveles de ruido, y enviar la información recopilada a una base de datos en la nube, la cual debe poder ser accedida por el público para la toma de decisiones turísticas informadas y por entidades como el MINAE y sus subdivisiones, y las municipalidades locales, para tener una mejor capacidad de manejo en el desarrollo sostenible y control ambiental en áreas clave del país, gracias a una futura red de sensado que haga disponibles y de fácil acceso los datos ambientales recolectados.

5.2.2. Síntesis del problema

No se cuenta con una solución de bajo costo para la medición de variables ambientales que permita contar con información actualizada para la gestión de ciudades y destinos turísticos inteligentes en Costa Rica.

5.3. Enfoque de la solución

Según las problemáticas descritas en el entorno y la definición del problema del proyecto, se tiene la oportunidad de desarrollar un sistema de sensado para la medición de parámetros de condiciones atmosféricas de calidad del aire, temperatura, humedad y niveles de contaminación acústica. Para esto, se pretende integrar una variedad de sensores utilizando un microcontrolador capaz de procesar los datos y ponerlos en línea para el acceso público de los mismos. Este sistema debe ser capaz de comunicaciones inalámbricas para enviar los datos recolectados a una base de datos en la web; para este fin, se debe implementar el nodo con un transmisor y un recibidor o central que se encargue de recibir y cargar los datos procesados a la nube.

Para la solución, se requiere integrar una variedad de sensores para la medición de parámetros atmosféricos del índice de calidad del aire (AQI), temperatura, humedad y ruido. Para medir AQI, se deben integrar sensores de detección de ozono (O_3), dióxido de nitrógeno (NO_2), dióxido de azufre (SO_2), monóxido de carbono (CO) y partículas PM2.5 y PM10; estos deben ser lo suficientemente sensibles para medir cuando estos se encuentran en concentraciones suficientes para ser consideradas nocivas para la salud.

Existen alternativas de menor costo, como lo son los sensores de compuestos orgánicos volátiles totales (TVOC) y dióxido de carbono equivalente (eCO_2), los cuales se pueden considerar si se desea un sistema de sensado de calidad del aire general. Los sensores de temperatura y humedad deben cubrir un amplio rango con buena precisión, mientras que el sensor de ruido debe tener un nivel de detección tal que pueda detectar cuando este sea nocivo para la salud humana.

Estos sensores deben ser integrados bajo un solo microcontrolador ESP32 que maneje la información recolectada y la pueda transmitir inalámbricamente a una central de recolección de datos; este también debe ser capaz de proveer alimentación a los sensores y tener puertos que sean compatibles con los protocolos de comunicación respectivos para cada uno. Tanto el transmisor como el receptor en el dúo nodo-central deben ser capaces de comunicaciones RF para la transmisión de datos de forma inalámbrica; además, la central debe también tener la capacidad de conexión a WiFi para la carga de datos a la nube.

Se busca que el sistema final permita realizar las mediciones de AQI, temperatura, humedad y nivel de ruido en un espacio abierto al aire libre. Los sensores deben tener la sensibilidad suficiente para poder detectar cuando los niveles de cada uno de los parámetros siendo medidos son lo suficientemente altos para considerarse nocivos para la salud. El sistema final estará compuesto por un dúo de microcontroladores: el nodo de sensado/transmisor junto con los distintos sensores, y la central/receptor recibiendo los datos obtenidos por los mismos. El sistema debe procesar los datos recolectados para que estos sean cargados a la web y sean de fácil acceso e interpretación para el usuario.

5.3.1. Descripción de la solución

El enfoque de esta solución es el diseño de un sistema que cumpla con los requerimientos descritos, siendo al mismo tiempo sencillo de integrar y replicar, intentando que no sea tan costoso. Para esto, los sensores deben cumplir con las necesidades de detección de gases nocivos, ya sea mediante un módulo dedicado a cada uno o un módulo de detección general, siempre que estos posean la capacidad de detectar cuando los niveles de gases peligrosos sean lo suficientemente altos para considerarse nocivos para la salud. Para esta opción se debe considerar también que los sensores pueden requerir calibración o un circuito de acondicionamiento adicional. Dicho esto, los sensores considerados para esta solución son: el módulo Adafruit ENS160, capaz de medir compuestos orgánicos volátiles totales (TVOC), dióxido de carbono equivalente (eCO_2) y medida general de índice de calidad del aire (AQI) basado en el estándar definido por la Agencia Ambiental Federal de Alemania (UBA); cuenta con puertos I^2C y SPI para facilidad de conexión y un algoritmo interno de compensación para humedad y temperatura; el sensor de partículas PMS5003; el sensor de ruido Gravity SEN0232; y el sensor Adafruit

Sensirion SHT40 de temperatura y humedad.

El microcontrolador ESP32 debe poseer la capacidad de enviar datos de manera inalámbrica por WiFi y por radiofrecuencia. Al estar utilizando varios módulos de sensado, el microcontrolador debe ser capaz de leer la trama de bits que estos le envían. Para el nodo y la central se considera el Heltec WiFi LoRa 32 V3. Este microcontrolador es compacto, eficiente y versátil; posee capacidad de comunicación WiFi, Bluetooth y LoRa; puertos UART, SPI, I^2C y GPIO con ADC para entradas analógicas, haciéndolo versátil a la hora de integrar componentes distintos. Posee salidas de 5 V y 3.3 V capaces de proveer hasta 500 mA de corriente, y tiene la capacidad de ser alimentado por batería. Esta última deberá ser fácilmente intercambiable y recargable; existen opciones disponibles como baterías LiPo de 3.7 V o bancos de poder capaces de proveer hasta 5 V.

La solución a implementar debe cumplir con los siguientes requerimientos del sistema:

- El sistema debe ser económicamente accesible de construir y replicar, sin comprometer la precisión de los datos obtenidos.
- Precisión adecuada para la detección de niveles peligrosos de gases nocivos en el aire, niveles de temperatura y humedad, además de nivel de ruido.
- El sistema debe ser simple de integrar, de fácil mantenimiento, capaz de soportar las condiciones climáticas del país y energéticamente eficiente para maximizar su longevidad.
- El sistema debe ser capaz de transmitir datos de manera inalámbrica entre el nodo y la central; el nodo no puede depender de comunicaciones WiFi, por lo que el microcontrolador debe ser compatible con comunicaciones en radiofrecuencia (RF) como LoRa.
- El sistema debe ser capaz de la carga de datos a la nube para la habilitación de información ambiental mediante la web.

Este se considera el mejor enfoque y estrategia de diseño para la solución diseñada dados los requerimientos del sistema, como la facilidad de integración de los componentes, la relativa viabilidad económica de la misma, la precisión de los sensores y la integración de comunicaciones RF en el microcontrolador.

6. Meta

Desplegar un sistema de sensado de bajo costo con amplia cobertura en todas las áreas pobladas y/o turísticas de Costa Rica, con el objetivo de modernizar el país como destino turístico inteligente, y proveer información y parámetros de calidad del aire, niveles de ruido y condiciones de temperatura y humedad al público, los gobiernos locales y los entes encargados del control ambiental en el país, como el Ministerio de Salud, el Instituto Meteorológico Nacional y el Ministerio de Ambiente y Energía y sus subdivisiones.

- Indicador: Se cuenta con mediciones geolocalizadas de calidad del aire, niveles de ruido, temperatura y humedad, disponibles al público, entes públicos y privados.

7. Objetivos

7.1. Objetivo general

- Desarrollo de un sistema de sensado de bajo costo capaz de medir calidad del aire general, nivel de ruido, temperatura y humedad, y hacer estos datos disponibles en la web para su fácil acceso.
    - Indicador: Se logra obtener datos de calidad del aire general, temperatura, humedad y ruido mediante la descarga de información desde la web, provista por el nodo de sensado desarrollado.

7.2. Objetivos específicos

1. Diseñar un nodo de sensado capaz de medir parámetros ambientales de calidad del aire general, temperatura, humedad y nivel de ruido.
    - Indicador: El nodo de sensado logra recolectar datos de calidad del aire general, temperatura, humedad y ruido, cumpliendo con los niveles de precisión y rango de detección que dictan las hojas de datos de los sensores.
2. Transmitir mediante comunicación LoRa los datos recolectados por el nodo de sensado a una central de recolección con pérdida mínima de datos.
    - Indicador: El nodo logra transmitir los datos recolectados por el nodo hacia la central con una pérdida mínima de datos, cumpliendo con límites de Relación Señal-Ruido (SNR) de -8 dB a +12 dB y de Indicador de Intensidad de Señal Recibida (RSSI) de -70 dBm a 0 dBm.
3. Procesar la información recibida por la central de recolección de datos y hacer esta información disponible al público mediante la carga de paquetes a una plataforma en la nube.
    - Indicador: Los datos son de fácil interpretación por parte del usuario, proporcionando información clara y actualizada; además, pueden ser cargados y descargados hacia y desde la web sin errores.

8. Fundamentos Teóricos

8.1. Calidad de aire y contaminantes atmosféricos.

La calidad del aire urbano se mide usualmente utilizando el Índice de Calidad del Aire (AQI o Air Quality Index). Este se rige bajo diferentes estándares dependiendo de la localidad; entre estos se encuentran la guía de la Organización Mundial de la Salud (OMS), el estándar de la Agencia de Protección Ambiental (EPA o Environmental Protection Agency), el Índice Centroamericano de Calidad de Aire (ICCA) y el estándar AQI-UBA definido por la Agencia Ambiental Federal de Alemania (Umweltbundesamt o UBA).

Por lo general, estos estándares indican que para la medición del AQI urbano se evalúan habitualmente contaminantes criterio: material particulado fino ($PM_{2,5}$) y grueso (PM_{10}), ozono (O_3), dióxido de nitrógeno (NO_2), dióxido de azufre (SO_2), monóxido de carbono (CO) y, en estándares como el AQI-UBA, se consideran los compuestos orgánicos volátiles totales (TVOC) y el dióxido de carbono equivalente (eCO_2). Estos parámetros se suelen expresar en $\mu g/m^3$ (CO también en mg/m^3), partes por millón (ppm) y partes por billón (ppb), y se reportan con promedios temporales definidos, ya sea a diario o cada cierta cantidad de horas, dependiendo del compuesto, dado que los efectos en la salud dependen tanto de la concentración como del tiempo de exposición.

8.1.1. Guía global de calidad del aire de la OMS.

La Organización Mundial de la Salud publicó en 2021 una actualización a su guía de calidad del aire (AQG). En esta se presentan guías y pautas sobre la medición correcta de contaminantes como referencia sanitaria para la gestión de la calidad del aire. Una de estas es el nivel promedio de contaminantes, observado en la Tabla 1; esta se utiliza como base para la definición del índice de calidad del aire en estándares internacionales.

Contaminante	**Tiempo promedio**	**AQG 2021**
$PM_{2,5}$	24 h	15 $\mu g/m^3$
PM_{10}	24 h	45 $\mu g/m^3$
O_3	8 h (máx. diario)	100 $\mu g/m^3$
O_3	8 h (temporada pico)	60 $\mu g/m^3$
NO_2	24 h	25 $\mu g/m^3$
SO_2	24 h	40 $\mu g/m^3$
SO_2	24 h	4 mg/m^3

Tabla 1: Niveles de exposición promedio recomendados de contaminantes en el aire [9].

8.1.2. Estándar de calidad de aire del Environmental Protection Agency o EPA

La Agencia de Protección Ambiental de los Estados Unidos ha desarrollado un índice estándar utilizado para el reporte de datos de calidad del aire. Este se basa en los contaminantes criterio del AQI definidos por la OMS y los niveles promedio de concentración definidos por la EPA, y se puede observar en la Tabla 2.

O_3 (ppm) 8h	**O_3 (ppm) 1h**	**$PM_{2,5}$ (µg/m^3) 24h**	**PM_{10} (µg/m^3) 24h**	**CO (ppm) 8h**	**SO_2 (ppb) 1h**	**NO_2 (ppb) 1h**	**AQI**	**Categoría**
0-54	-	0.0-9.0	0-54	0-4.4	0-35	0-53	0-50	Bueno
55-70	-	9.1-35.4	55-154	4.5-9.4	36-75	54-100	51-100	Moderado
71-85	125-164	35.5-55.4	155-254	9.5-12.4	76-185	101-360	101-150	Dañino para grupos sensibles
86-105	165-204	55.5-125.4	255-354	12.5-15.4	186-304	361-649	151-200	Dañino a la salud
106-200	205-404	125.5-225.4	355-424	15.5-30.4	304-604	650-1,249	201-300	Muy dañino a la salud
201+	405-604	225.5-325.4	425-604	30.5-50.4	605-1004	1,250-2,049	301-500	Peligroso

Tabla 2: Tabla de calidad de aire de la EPA [31].

8.1.3. Índice Centroamericano de Calidad del Aire o ICCA

Al igual que la EPA, en Centroamérica el ICCA se basa en la guía de la OMS para definir su propio índice, pero utiliza diferentes estándares de niveles para su calificación según sus promedios; este se puede observar en la Tabla 3.

O_3 (ppm) 8h	**$PM_{2,5}$ (µg/m^3) 24h**	**PM_{10} (µg/m^3) 24h**	**CO (ppm) 8h**	**SO_2 (ppm) 24h**	**NO_2 (ppm) 24h**	**AQI**	**Categoría**
0-0.064	0-15.3	0-55	0-4.4	0-0.034	-	0-50	Bueno
0.065-0.084	15.5-40.2	56-154	4.5-9.4	0.035-0.144	-	51-100	Moderado
0.085-0.104	40.5-65.4	155-254	9.5-12.4	0.145-0.224	-	101-150	Dañino para grupos sensibles
0.105-0.124	66-159	255-354	12.5-15.4	0.225-0.304	-	151-200	Dañino a la salud
0.125-0.373	160-250	355-424	15.5-30.4	0.305-0.604	0.452-1.24	201-300	Muy dañino a la salud
¿0.374	251-500	425-604	30.5-50.4	0.605-1.005	1.25-2.04	301-500	Peligroso

Tabla 3: Tabla de calidad de aire de la ICCA [33].

8.1.4. Estándar AQI-UBA de la UBA.

Este estándar de calidad del aire es derivado de la guía de la Agencia Ambiental Federal de Alemania (UBA), basado en niveles promedio de TVOC y eCO_2. Aunque es más utilizado localmente en Alemania, este estándar ha sido referenciado y adoptado por varios países y organizaciones. Su índice es observable en la Tabla 4.

eCO_2 (ppm)	**TVOC (mg/m^3)**	**TVOC (ppm)**	**AQI**	**Categoría**	**Límite de exposición**
400-600	0.3	0-0.065	1	Excelente	Sin límite
600-800	0.3-1	0.065-0.22	2	Bueno	Sin límite
800-1000	1-3	0.22-0.65	3	Moderado	<12 meses
1000-1500	3-10	0.65-2.2	4	Malo	<1 mes
1500	10-25	2.2-5.5	5	Insalubre	Horas

Tabla 4: Tabla de calidad de aire de la UBA [39].

8.1.5. Contaminantes y condiciones atmosféricas

Los contaminantes atmosféricos y las condiciones ambientales son factores determinantes de la calidad de vida y la salud pública del ser humano. Su monitoreo permitiría identificar riesgos y peligros, diseñar mejores políticas ambientales, tomar decisiones informadas y mejorar la disponibilidad de información ambiental para el público. En 2021 la OMS señaló que la exposición a partículas finas ($PM_{2,5}$) fue responsable de millones de muertes prematuras en 2019, y que los efectos del ruido ambiental y del estrés térmico también representan riesgos significativos para la población [32].

- $PM_{2,5}$ y PM_{10}

  La materia particulada o PM sólida o líquida proviene mayormente de los procesos de combustión y se encuentra suspendida en el aire. $PM_{2,5}$ ($\leq$ 2.5 µm) penetra en los alvéolos y pasa al torrente sanguíneo; se asocia a enfermedades cardiovasculares, cáncer de pulmón y mortalidad prematura. PM_{10} ($\leq$ 10 µm) afecta vías respiratorias superiores, generando tos, bronquitis y asma.

- O_3

  El ozono troposférico es un oxidante secundario que se forma a partir de óxidos de nitrógeno (NO_X) y compuestos orgánicos volátiles (VOC) reaccionando químicamente bajo radiación solar. Sus principales fuentes son contaminantes emitidos por automóviles, plantas industriales y químicos. Causa irritación, disminuye la función respiratoria y puede generar o agravar el asma y causar daño pulmonar [11].

- NO_2

  El dióxido de nitrógeno es un contaminante derivado principalmente del tráfico vehicular y el humo del tabaco. Puede causar inflamación de los bronquios, disminución de la función pulmonar y aumento de la susceptibilidad a infecciones respiratorias.

- SO_2

El dióxido de azufre es un contaminante derivado de la quema de combustibles fósiles, capaz de inducir asma con síntomas agudos y dificultar la respiración. El SO_2 provoca la formación de trióxido de azufre (SO_3); estas partículas contribuyen a la contaminación de partículas PM [14].

- CO

  El monóxido de carbono es un gas tóxico, invisible e inodoro, que limita el transporte de oxígeno en sangre y puede causar la muerte en altas concentraciones. Es un producto de la combustión de hidrocarburos como gas, gasolina, queroseno, carbón, petróleo o madera [10].

- VOC

  Los compuestos orgánicos volátiles o VOC son gases emitidos por compuestos sólidos o líquidos de origen orgánico, como benceno, formaldehído, tolueno y tetracloroetileno. Provienen de fuentes como emisiones vehiculares, pinturas y aerosoles, humo del tabaco, químicos de limpieza y pesticidas. Pueden causar efectos nocivos para la salud como dolores de cabeza agudos, irritación en los ojos y el sistema respiratorio, náuseas y daños al hígado, riñones y sistema nervioso central [17].

- eCO_2

  El dióxido de carbono equivalente es una unidad estandarizada que mide el impacto que un gas de efecto invernadero tendría en relación con su equivalente en dióxido de carbono (CO_2). Entre estos gases se encuentran el metano (CH_4), el CO_2 y el NO_2, entre otros. Estos pueden causar efectos como dolores de cabeza, fatiga y problemas respiratorios [18].

- Ruido

  La definición tradicional de ruido es: "sonido no deseado que interfiere con actividades humanas e interfiere con la calidad de vida", es decir, cualquier sonido indeseado o perjudicial que interfiera con el descanso, la comunicación o la concentración. A diferencia de los contaminantes químicos, el ruido no deja residuos, pero sus efectos fisiológicos y psicológicos son acumulativos y medibles.

  La contaminación por ruido, o acústica, se considera peligrosa a partir de 85 dB, con efectos como dolores de cabeza, estrés y alta presión sanguínea. Sonidos de 120 dB pueden causar daños más directos, como pérdida auditiva [20].

  En Costa Rica la Procuraduría General de la República indica los límites de ruido en el país, observables en la Tabla 5.

Zona	**Límite diurno (dBA)**	**Límite nocturno (dBA)**
Residencial	65	45
Comercial	70	55
Industrial o Agrícola	70	60
Tranquilidad	50	45
Mixta	70	45

Tabla 5: Límites de ruido en Costa Rica [5].

- Temperatura y Humedad Relativa.

  El ozono troposférico, también denominado *"smog"*, está vinculado directamente con el cambio climático y la salud pública. Altos niveles de temperatura y humedad aceleran las reacciones químicas que forman O_3 cuando los VOC y NO_X interactúan con la radiación solar. El aumento en la temperatura provoca que las concentraciones de ozono incrementen significativamente, y la alta humedad ayuda a diluir contaminantes en la atmósfera más rápido [24].

  "...las altas temperaturas han disparado la contaminación por ozono troposférico durante la primavera de 2025, alcanzando niveles inusuales para la época... con episodios de contaminación asociados directamente a las altas temperaturas registradas este año." [24]

  A continuación se presentan los niveles y tiempos de exposición a los contaminantes y condiciones atmosféricas, observables en la Tabla 6.

Parámetro	**Tiempo de exposición**	**Valor guía (OMS/EPA/UE)**
$PM_{2,5}$	24 h	15 $\mu g/m^3$
$PM_{2,5}$	Anual	5 $\mu g/m^3$
PM_{10}	24 h	45 $\mu g/m^3$
PM_{10}	Anual	15 $\mu g/m^3$
O_3	8 h máx. diario	100 $\mu g/m^3$
NO_2	24 h	25 $\mu g/m^3$
NO_2	Anual	10 $\mu g/m^3$
SO_2	24 h	40 $\mu g/m^3$
CO	24 h	4 mg/m^3
TVOC	Exposición continua	0.5 mg/m^3 (exteriores)
eCO_2	Exposición continua	$\leq$ 1000 ppm (con ventilación)
Ruido	Día	$\leq$ 65 dB
Ruido	Noche	$\leq$ 45 dB
Temperatura	Exposición crítica	$\geq$ 35 °C (riesgo de golpe de calor); $<$ 10 °C (riesgo de hipotermia)
Humedad relativa	Exposición continua	30–60 % (fuera de este rango aumentan riesgos de salud y proliferación de moho)

Tabla 6: Niveles y tiempos recomendados de exposición a contaminantes y condiciones atmosféricas.

8.2. Microcontrolador ESP32

ESP32 es una familia de sistemas en chip (System on Chip o SoC) desarrollada por Espressif Systems, diseñada específicamente para aplicaciones que requieren conectividad inalámbrica y procesamiento eficiente. Este dispositivo se ha convertido en una de las plataformas más populares en el ámbito de la electrónica y el Internet de las Cosas (IoT) gracias a su bajo costo, alto rendimiento y versatilidad [36]. Entre sus características más destacadas se encuentran:

- **Conectividad:** Incluye Wi-Fi 802.11, Bluetooth 4.2 (clásico y BLE) y en ocasiones comunicaciones por radiofrecuencia (LoRa), lo que permite la comunicación inalámbrica con otros dispositivos y redes.
- **Procesador de doble núcleo:** Basado en la arquitectura Xtensa LX6, puede alcanzar hasta 240 MHz, lo que le permite ejecutar tareas complejas en tiempo real.
- **Amplia capacidad de entrada/salida:** Dispone de pines GPIO, con soporte para protocolos ADC, DAC, PWM, SPI, I^2C, UART, entre otros.
- **Memoria y almacenamiento:** Cuenta con buenas capacidades de memoria

SRAM y memoria flash externa.

- **Bajo consumo energético:** Ideal para dispositivos alimentados por batería, gracias a sus modos de suspensión y gestión energética eficiente.

El ESP32 es compatible con múltiples entornos de desarrollo, como Arduino IDE, PlatformIO y ESP-IDF, lo que facilita su adopción por estudiantes, desarrolladores y profesionales. Su capacidad para integrar conectividad, procesamiento y expansión en un solo chip lo convierte en una solución robusta para proyectos de domótica, automatización industrial, monitoreo remoto y sistemas embebidos.

8.3. Comunicaciones LoRa

LoRa (Long Range o larga distancia) es una tecnología de modulación de radiofrecuencia diseñada para comunicaciones de largo alcance y bajo consumo energético. Utiliza un tipo especial de modulación de espectro ensanchado llamada Chirp Spread Spectrum (CSS), un ejemplo de transmisión con este tipo de modulación de puede observar en la Figura 1. La información se codifica mediante "chirps", que son señales cuya frecuencia aumenta o disminuye de manera lineal con el tiempo. Este método de modulación permite que cada bit de información se distribuya a lo largo de un amplio rango de frecuencias, lo que mejora la resistencia a interferencias y la detección de señales muy débiles, y permite transmitir datos a distancias largas con bajo consumo de energía [15].

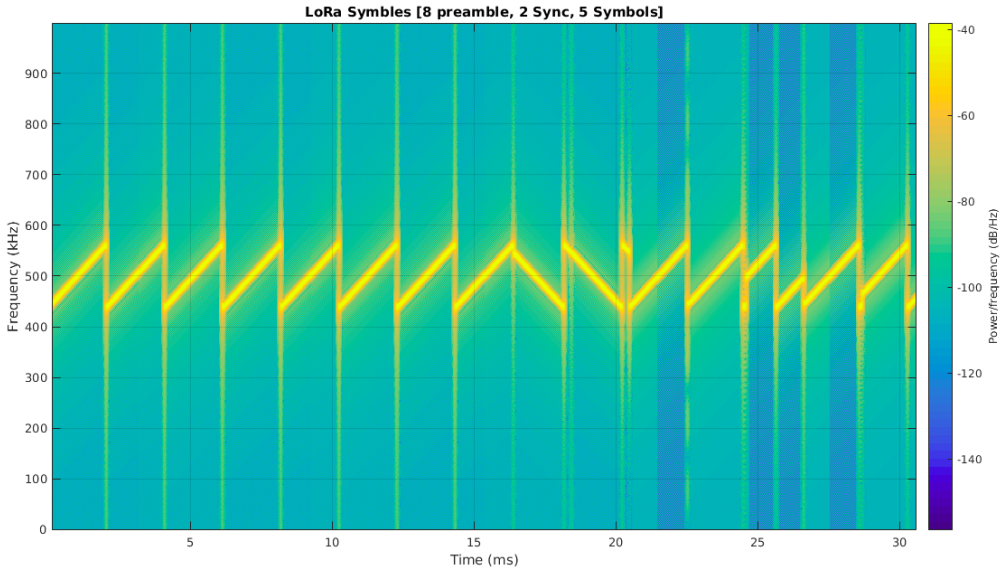

Figura 1: Ejemplo de mensaje LoRa. [15]

8.3.1. Legislación Costarricense

En Costa Rica cualquier persona puede operar un sistema LoRa, se recomienda obtener una licencia de radioaficionado. Estas licencias se clasifican de acuerdo con las siguientes categorías:

- Novicio (Clase C).
- Intermedio (Clase B).
- Superior (Clase A).

LoRa opera en las bandas de 433.05 MHz a 434.79 MHz y 902 MHz a 928 MHz en Costa Rica [25]. La restricción de PIRE (Potencia Isotrópica Radiada Equivalente) para estas bandas depende del tipo de licencia que se posea:

- Novicio: 200 W.
- Intermedio: 1000 W.
- Superior: 1995 W.

8.3.2. Relación Señal-Ruido (Signal-Noise Ratio o SNR)

El SNR es una métrica que indica la calidad de una señal comparando su potencia frente al nivel de ruido presente en el canal de comunicación. En términos prácticos, un valor de SNR alto significa que la señal se distingue claramente del ruido, mientras que un valor bajo indica que el ruido interfiere significativamente en la transmisión de datos. En sistemas que utilizan comunicaciones inalámbricas como LoRa, Wi-Fi o Bluetooth, el SNR permite evaluar la fiabilidad del enlace; generalmente, un SNR positivo (por ejemplo, entre +5 dB y +20 dB) indica una señal buena o excelente, mientras que valores cercanos a 0 dB o negativos implican una comunicación débil o inestable. Por ejemplo, un SNR de 10 dB se considera aceptable para la mayoría de las aplicaciones IoT, ya que garantiza una recepción adecuada con pocos errores. En resumen, cuanto mayor sea el SNR, mejor será la calidad y estabilidad de la señal recibida [26].

8.3.3. Indicador de Intensidad de Señal Recibida (Received Signal Strength Indicator o RSSI)

El RSSI es un parámetro que permite medir la potencia con la que una señal inalámbrica es recibida luego de considerar las pérdidas de propagación, por interferencias y por obstáculos en el entorno. En general, entre más cercano sea su valor a 0 dBm, es indicador de una señal fuerte, mientras que valores muy negativos reflejan una recepción débil. Por ejemplo, un RSSI de –40 dBm se considera excelente, mientras que uno de –90 dBm o inferior suele considerarse malo.

Para interpretar la calidad de una conexión, el RSSI proporciona una referencia rápida del alcance y estabilidad del enlace. En la práctica, una señal entre –30 dBm y –50 dBm se considera excelente; de –50 dBm a –70 dBm es buena y funcional para la mayoría de aplicaciones, mientras que de –70 dBm en adelante ya pueden existir pérdidas de paquetes o problemas de conexión. Es importante notar que el RSSI solo mide la potencia de la señal recibida, no su estabilidad frente a ruido o interferencia, por esto se recomienda analizarlo junto con otras métricas como el SNR. Por ejemplo, un dispositivo puede tener un RSSI alto, pero si el SNR es bajo, la comunicación puede ser ineficiente, o también se puede tener un buen valor de RSSI que, en combinación con un SNR adecuado, permite concluir que se tiene una transmisión confiable y de alta calidad [27].

8.3.4. Factor de dispersión (Spreading factor o SF)

El factor de dispersión es un parámetro en comunicaciones LoRa que controla la velocidad de una transmisión al definir cuántos bits se codifican por símbolo. Este puede variar entre SF7 y SF12 y determina la duración de cada símbolo transmitido. Un valor de SF7 implica una transmisión más rápida pero menos resistente al ruido y con menor alcance, mientras que un valor de SF12 conlleva una transmisión más lenta pero mucho más robusta y capaz de cubrir distancias más largas. Sin embargo, este aumento en alcance viene acompañado de una reducción considerable en la tasa de datos que pueden ser enviados y un mayor consumo energético del sistema LoRa, ya que el tiempo de transmisión se duplica con cada incremento de este parámetro [28].

Al aumentar el SF también se obtiene mayor sensibilidad de recepción, lo que significa que se pueden recibir señales con valores de RSSI y SNR más bajos manteniendo una comunicación confiable; en cambio, con un SF bajo es necesario tener valores más altos de estos parámetros para lograr buena recepción. Un buen enlace LoRa no depende solo de tener un alto RSSI o SNR, sino de una combinación adecuada del SF con las condiciones de donde se está localizando el sistema.

En la Tabla 7 se presentan las velocidades y tamaño máximo de paquete que se puede enviar por nivel de factor de dispersión.

Factor de dispersión	**Ancho de banda del canal**	**Tasa de bits**	**Tamaño del paquete**
SF12	125 kHz	250 bps	51 bytes
SF11	125 kHz	440 bps	51 bytes
SF10	125 kHz	980 bps	51 bytes
SF9	125 kHz	1.76 kbps	115 bytes
SF8	125 kHz	3.13 kbps	242 bytes
SF7	125 kHz	5.47 kbps	242 bytes

Tabla 7: Factor de dispersión y velocidades de transmisión. [29].

8.4. Comunicaciones WiFi

WiFi, según el estándar IEEE 802.11, es una tecnología de red inalámbrica usada para conectar dispositivos a internet [35]. En el contexto de este proyecto y dispositivos IoT en general, el WiFi permite la transmisión de datos a un *gateway* o directamente a un servidor en la nube.

Los dispositivos IoT con acceso a WiFi usualmente operan en la banda de 2.4 GHz debido a su capacidad de penetración de obstáculos y amplia compatibilidad con otros dispositivos. En caso de ser necesario un mayor ancho de banda para la transmisión de datos y de tener la disponibilidad, se puede utilizar la red de 5 GHz, aunque esta posee menor alcance y capacidad de penetración.

Muchos chips modernos, como los utilizados en microcontroladores ESP32, poseen capacidades WiFi con bajo consumo de potencia y modos de suspensión para conservar energía.

8.5. Protocolo I^2C

El protocolo Inter-Integrated Circuit (I^2C) es un estándar de comunicación serial síncrona que permite la conexión de dispositivos mediante solo dos líneas de comunicación, una para datos (Serial Data Line o SDA) y otra para reloj (Serial Clock Line o SCL) [21].

El protocolo I^2C utiliza una arquitectura maestro-esclavo, observable en la figura 2, en la cual el maestro controla el flujo de datos y genera las señales de reloj, mientras que los dispositivos esclavos responden cuando detectan su dirección específica. El intercambio de información se realiza mediante condiciones de inicio y parada, transmisión de datos y confirmación de recepción [21].

Este protocolo admite distintas velocidades de operación, siendo las más comunes 100 kHz (modo estándar) y 400 kHz (modo rápido), lo cual es suficiente para la comunicación entre componentes en sistemas embebidos y dispositivos IoT.

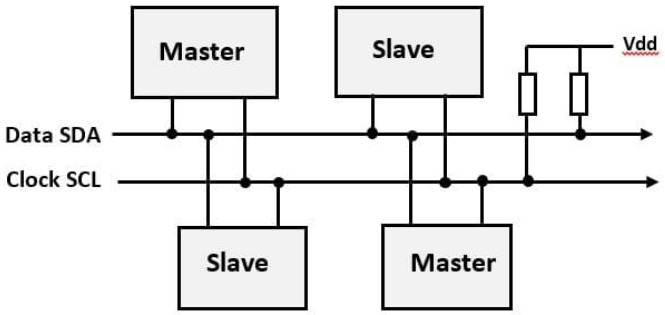

Figura 2: Diagrama de bloques del funcionamiento del protocolo I^2C [22].

8.6. Protocolo UART

Transmisor-Receptor Asíncrono Universal (Universal Asynchronous Receiver-Transmitter o UART) como su nombre lo implica es un protocolo de comunicación en serie asíncrono. A diferencia de otros protocolos, UART no utiliza una señal de reloj compartida, sino que el transmisor y el receptor operan con la misma velocidad de transmisión o *"baud-rate"* [7], observable en la figura 3. La comunicación se realiza mediante una trama que incluye un bit de inicio, los bits de datos, un bit opcional de paridad y uno o más bits de parada.

Este formato permite una transmisión punto a punto simple y confiable entre dispositivos. Utiliza una línea de transmisión (TX) y una de recepción (RX). Su facilidad de implementación, alta compatibilidad y estabilidad lo convierten en uno de los protocolos más utilizados en sistemas embebidos y dispositivos IoT [7].

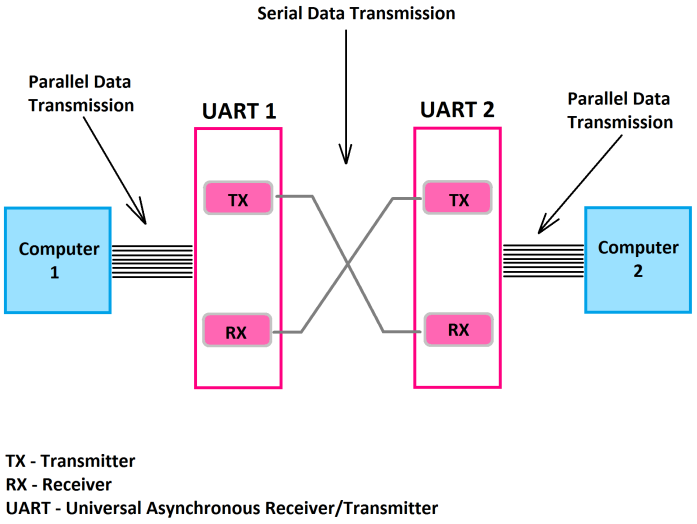

Figura 3: Diagrama de bloques del funcionamiento del protocolo UART [23].

8.7. Base de datos en la nube

Una base de datos en la nube es un servicio de almacenamiento, manejo y consulta de datos que se ejecuta sobre una infraestructura computacional en la nube, en lugar de un servidor local. Estas bases de datos son administradas por proveedores de nube (como Google Cloud, AWS, Adafruit IO o Azure) y ofrecen alta disponibilidad, escalabilidad, redundancia y acceso remoto. Estas características son esenciales para sistemas IoT, donde decenas o cientos de dispositivos distribuidos generan información constantemente [8].

Estas pueden clasificarse en relacionales, no relacionales y orientadas a series temporales [8]. Para el caso de este proyecto, las bases de datos de tipo *time-series* resultan particularmente adecuadas, ya que los registros de datos obtenidos por los sensores pueden ser indexados en función del tiempo y permiten análisis de tendencias o comportamiento histórico. La transmisión de información hacia una base de datos en la nube puede realizarse mediante diversos protocolos de comunicación como Hypertext Transfer Protocol (HTTP), Constrained Application Protocol (CoAP) y Message Queuing Telemetry Transport (MQTT).

8.7.1. Message Queuing Telemetry Transport o MQTT

El protocolo MQTT es un estándar de comunicación ligero basado en un modelo publicador–suscriptor. Fue diseñado para el intercambio eficiente de mensajes entre dispositivos con recursos limitados o conexiones de red inestables. Utiliza un servidor central llamado publicador que gestiona los mensajes entre los clientes, permitiendo que los

sensores envíen datos específicos, mientras los suscriptores reciben solo la información de su interés. Este protocolo se considera el más conveniente para este proyecto, ya que ofrece una comunicación ligera, bajo consumo de ancho de banda y es altamente eficiente, permitiendo garantizar una buena entrega de datos [16].

8.8. Sensores de calidad de aire

Los sensores de calidad de aire permiten medir la presencia y concentración de contaminantes atmosféricos en entornos urbanos o naturales, proporcionando información esencial para la evaluación ambiental y la protección de la salud pública.

Los contaminantes más comúnmente monitoreados incluyen el material particulado ($PM_{2,5}$ y PM_{10}), O_3, NO_2, SO_2 y el CO. A estos se suman variables complementarias como los TVOC y el eCO_2. Los sensores varían según el tipo de contaminante: electroquímicos para gases, ópticos para partículas y semiconductores para compuestos volátiles [9].

8.9. Sensores de temperatura y humedad relativa

Los sensores de temperatura y humedad relativa constituyen componentes esenciales en los sistemas de monitoreo ambiental y de calidad del aire; ambos parámetros influyen significativamente en la dinámica de los contaminantes atmosféricos y en la percepción de confort ambiental. La temperatura afecta la formación de ozono troposférico, ya que acelera las reacciones fotoquímicas entre los óxidos de nitrógeno (NO_X) y los VOC, mientras que la humedad relativa influye en la dispersión de gases, su concentración y el comportamiento del material particulado suspendido en el aire [24].

Dado esto, los sistemas de sensado de calidad del aire suelen incluir sensores combinados de temperatura y humedad, no solo para proporcionar información atmosférica, sino también porque las respuestas de los sensores electroquímicos y semiconductores para gases dependen de la temperatura y la humedad ambiental. Medir ambas variables permite aplicar compensaciones y correcciones en los cálculos de concentración, mejorando la precisión del sistema de sensado.

8.10. Sensores de nivel de ruido

Los sensores de ruido permiten medir los niveles de presión sonora (SPL) y cuantificar la exposición acústica de una zona, expresada comúnmente en decibelios ponderados (dBA). Estos sensores se integran con microcontroladores que procesan las señales y calculan el nivel equivalente de ruido.

Incluirlos en redes de monitoreo ambiental podría complementar la información obtenida por un sistema de monitoreo ambiental, ofreciendo una visión más completa de la calidad de vida urbana, ya que la medición del ruido permite evaluar la calidad del entorno desde una perspectiva de bienestar y confort.

8.11. Baterías de litio tipo Li-Po

Las baterías de polímero de litio (Li-Po), a diferencia de las más tradicionales de ión de litio (Li-Ion), utilizan un electrolito en forma de gel o polímero sólido, en lugar de un electrolito líquido. Esto les permite tener mayor flexibilidad de forma, menor peso y mayor densidad energética. Están formadas por una o más celdas (típicamente de 3.7 V) con capacidad variable según el tamaño. Usualmente requieren de un módulo de protección para evitar sobrecargas, sobredescargas o cortocircuitos [13].

Los bancos de poder son dispositivos modernos de almacenamiento y entrega de energía portátiles que integran celdas, ya sea de Li-Ion o Li-Po, junto con protección interna y convertidores que elevan el voltaje de salida a 5 V USB estándar. Para aplicaciones donde una batería de 3.7 V no es suficiente, un banco de poder puede funcionar como una alternativa, siempre y cuando el dispositivo que lo va a utilizar posea la capacidad de ser conectado a este.

9. Especificaciones Técnicas

Para el desarrollo del sistema propuesto en este proyecto se hizo uso de software y hardware específico; a continuación, se presentan sus características para un mejor entendimiento del sistema y del proceso de diseño e integración.

9.1. Hardware

9.1.1. Módulo de Sensado Adafruit ENS160

Este componente es un módulo de sensado de calidad de aire general que contiene el sensor ENS160 y su circuito de acondicionamiento integrados en una sola placa para facilidad de integración. En general, este módulo es versátil y económico; posee un amplio rango de detección de compuestos orgánicos volátiles (TVOC) y gases contaminantes generales (eCO_2), haciéndolo capaz de identificar cuando los niveles de cualquiera de estos son peligrosos para la salud humana con una precisión adecuada y darle una clasificación basada en el estándar AQI-UBA. Posee un consumo energético bajo, dejando más que suficiente potencia para que otros componentes puedan ser alimentados también; además, este es compatible con los protocolos de comunicación SPI e I^2C, haciendo que la interfaz entre este y el microcontrolador a utilizar sea simple y eficiente.

Se puede observar el módulo Adafruit ENS160 en la Figura 4.

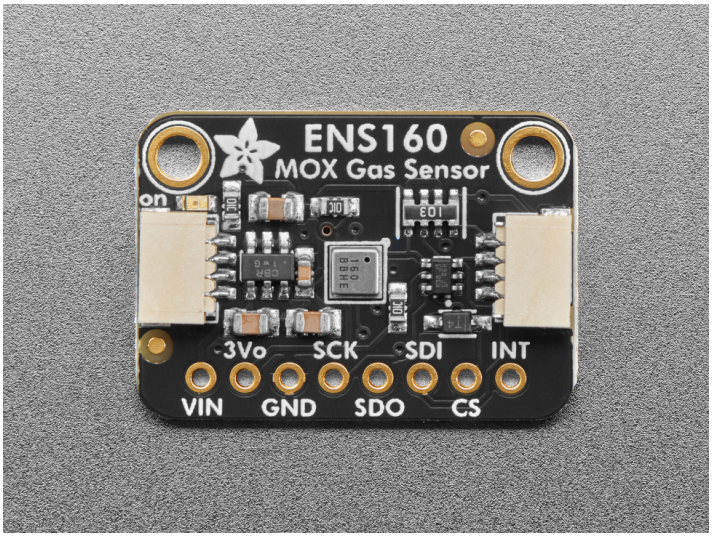

Figura 4: Adafruit ENS160.

En la Tabla 8 se pueden observar las características eléctricas más relevantes del módulo.

Parámetro	**Valor**
Alimentación	3.3 a 5 V
Consumo de corriente	$\sim$ 29 mA
Temperatura de operación	-40 °C a 85 °C
Tiempo de calentamiento	$\sim$ 3 min
Tiempo de respuesta	$\sim$ 3 s

Tabla 8: Características del módulo Adafruit ENS160 [40].

Seguidamente, se presentan las características de sensado del módulo en las Tablas 9 y 10.

Sensado de TVOC	
Parámetro	**Valor**
Rango de detección	0 a 65000 ppb
Precisión	$\pm$15 ppb o $\pm$15 %, el que sea mayor

Tabla 9: Características de sensado de TVOC para el Adafruit ENS160 [40].

Sensado de eCO_2	
Parámetro	**Valor**
Rango de detección	400 a 65000 ppm
Precisión	$\pm$50 ppm o $\pm$15 %, el que sea mayor

Tabla 10: Características de sensado de eCO_2 para el Adafruit ENS160 [40].

9.1.2. Módulo de Sensado Adafruit SHT40

Este es un módulo de sensado de bajo costo para temperatura y humedad. No solo posee la capacidad de detectar estos parámetros con excelente precisión, sino que también viene calibrado de fábrica y utiliza una interfaz I^2C para fácil integración con el microcontrolador ESP32 seleccionado.

Se puede observar el módulo Adafruit SHT40 en la Figura 5.

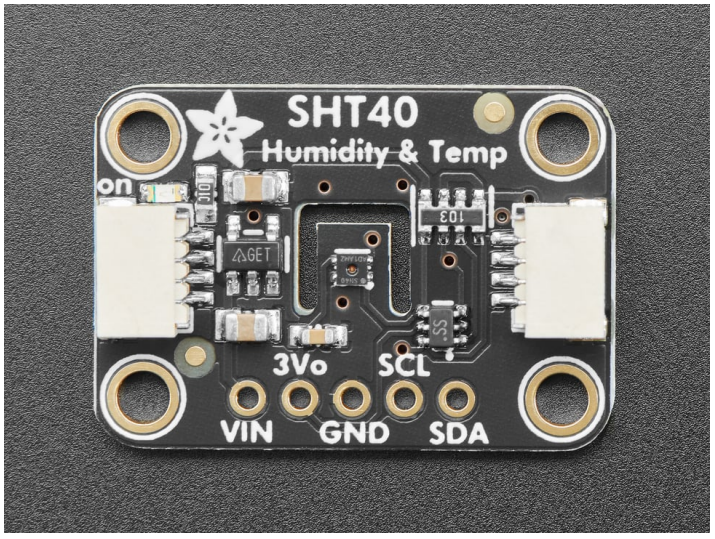

Figura 5: Adafruit SHT40.

Este módulo de sensado de temperatura y humedad posee las características presentadas en la Tabla 11.

Parámetro	**Valor**
Alimentación	3.3 a 5 V
Consumo de corriente	0.4 µA en promedio
Rango de sensado de temperatura	-40 °C a 85 °C (±0.2 °C)
Rango de sensado de humedad relativa	0 a 100 % RH (±1.8 % RH)
Tiempo de respuesta	∼ 6.9 ms

Tabla 11: Características del módulo Adafruit SHT40 [41].

9.1.3. Sensor de partículas Plantower PMS5003

Este sensor de material particulado se eligió debido a su bajo costo, bajo consumo energético, buen rango de detección y alta precisión. Este utiliza una interfaz de comunicación UART, lo cual lo hace de fácil integración con el microcontrolador seleccionado. Puede observarse en la Figura 6.

Figura 6: Plantower PMS5003.

Las características eléctricas y de sensado del sensor se pueden observar en la Tabla 12.

Parámetro	**Valor**
Alimentación	5 V
Consumo de corriente	100 mA en promedio
Temperatura de operación	-10 °C a 60 °C
Rango de tamaño de partículas	0.3 a 10 µm ($PM_{1,0}$, $PM_{2,5}$, PM_{10})
Rango de detección	0 a 500 µg/m^3 ±10 µg/m^3 o ±10 %, el que sea mayor
Tiempo de respuesta	∼ 1 s

Tabla 12: Características del sensor Plantower PMS5003 [44].

9.1.4. Sensor de ruido Gravity SEN0232

Según la Organización Mundial de la Salud (OMS), el rango de nivel de ruido peligroso para el ser humano es de 75 a 120 dBA, haciendo al sensor SEN0232 perfecto para la detección de contaminación acústica gracias a su rango de detección y alta precisión. Su

bajo consumo de corriente también lo hace apto para integrarse con los otros módulos de sensado que poseen un mayor consumo de potencia. Además, el ESP32 posee varios pines de entrada analógica a digital (ADC), haciendo que su salida analógica siga siendo apta para su uso.

Se puede apreciar el módulo en la Figura 7.

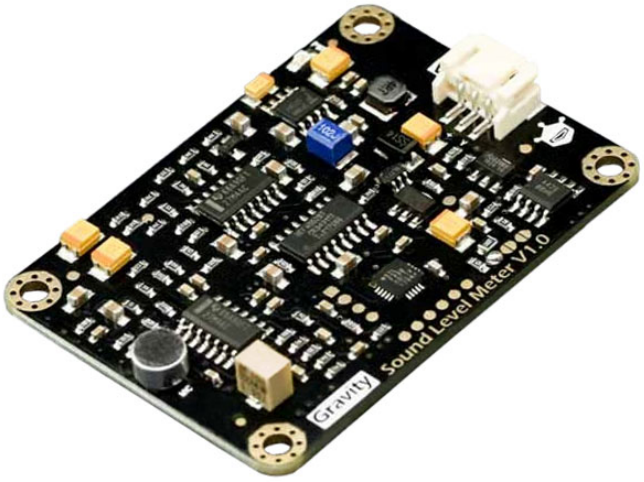

Figura 7: Gravity SEN0232.

Este compacto sensor de ruido posee las características eléctricas y de sensado observables en la Tabla 13.

Parámetro	**Valor**
Alimentación	3.3 a 5 V
Consumo de corriente	22 mA a 3.3 V o 14 mA a 5 V
Salida analógica	0.6 a 2.6 V
Rango de detección	30 a 130 dBA
Precisión	$\pm$1.5 dB
Tiempo de respuesta	$\sim$ 125 ms

Tabla 13: Características del sensor SEN0232 [43].

9.1.5. HELTEC WiFi LoRa 32(V3) ESP32S3 + SX1262

Este es un microcontrolador ESP32 versátil y robusto, especialmente de preferencia para aplicaciones IoT, ciudades inteligentes, control industrial y aplicaciones inalámbricas. Tiene un procesador de doble núcleo a 240 MHz, haciéndolo capaz de manejar y procesar una gran cantidad de instrucciones, datos y acciones sin problema. Además, posee un gran número de puertos con protocolos de comunicación variados como convertidores analógico-digital (ADC), Universal Asynchronous Receiver/Transmitter (UART), circuito inter-integrado (I^2C) e interfaz periférica serial (SPI), resultando más que apto para la integración de los módulos de sensado descritos anteriormente. [37]

Posee capacidades de comunicación inalámbrica por WiFi 802.11 b/g/n, Bluetooth de baja energía (Bluetooth Low Energy o BLE) y comunicaciones LoRa a 433 MHz y 902 MHz a 928 MHz, haciéndolo apto para la transmisión de datos entre dos de estos microcontroladores y la carga de datos a la web. Tiene dos salidas de alimentación a 3.3 V y una a 5 V, y puede ser alimentado por una simple batería de 3.7 V, haciéndolo capaz de proveer potencia a los módulos de sensado requeridos y no depender de alimentación por USB. Este microcontrolador puede observarse en la Figura 8.

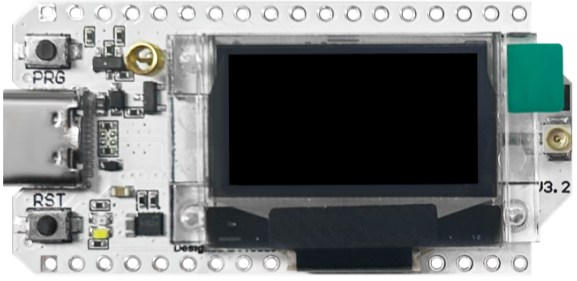

Figura 8: Heltec WiFi Lora 32 V3.

Este microcontrolador posee las características eléctricas observables en la Tabla 14.

Parámetro	**Valor**
Alimentación	3.7 V (Batería) a 5 V (USB)
Consumo de corriente	$\sim$ 150 mA promedio o $\sim$ 250 mA WiFi/LoRa activo
Voltaje de salida	3.3 V (Batería) a 5 V (USB)
Corriente de salida	500 mA
Frecuencia de RF	433 MHz y 902 MHz a 928 MHz
Potencia de transmisión (LoRa)	hasta 21 $\pm$1 dBm o 125 mW
Sensibilidad de recepción	-136 dBm (SF12, BW=125 kHz)
Procesador	ESP32-S3 (dual-core, 240 MHz)

Tabla 14: Características del microcontrolador Heltec WiFi LoRa v3 [37].

9.1.6. Inui BI-B41

Este es un banco de poder de la marca Inui, observable en la Figura 9 seleccionado debido a su capacidad de carga y su interfaz USB a USB-C, capaz de proveer alimentación de 5 V al nodo de sensado, esta última característica siendo necesaria para que se pueda utilizar el pin de alimentación de 5 V del microcontrolador y dar potencia al sensor PMS5003. A continuación se presentan las características eléctricas del BI-B41.

Figura 9: Banco de poder Inui BI-B41.

En la Tabla 15 se pueden observar las características de este componente.

Parámetro	**Valor**
Voltaje de entrada	5 V
Corriente de entrada	3 A
Voltaje de salida	5 V
Corriente de salida	3 A
Capacidad	10 000 mAh o 37 Wh

Tabla 15: Características del Banco de poder Inui BI-B41 .

9.2. Software

9.2.1. Arduino IDE

Para el desarrollo y programación del nodo de sensado ambiental se seleccionó el entorno Arduino IDE debido a su alta compatibilidad con el microcontrolador Heltec WiFi LoRa 32 V3 y con los sensores del sistema desarrollado en el proyecto. Este entorno ofrece una interfaz sencilla e intuitiva que facilita la escritura, compilación y carga del código al microcontrolador, además de contar con una amplia disponibilidad de librerías específicas para los sensores del sistema, lo que agiliza el proceso de programación e integración. Asimismo, Arduino IDE proporciona herramientas útiles como el monitor serial, que permite observar en tiempo real el comportamiento de las variables medidas y verificar el funcionamiento del sistema. Dado su carácter multiplataforma, el sólido soporte de la comunidad y su compatibilidad con servicios en la nube, el uso de Arduino IDE resulta una opción práctica, accesible y eficiente para el desarrollo de sistemas de sensado dentro de entornos urbanos inteligentes.

9.2.2. Adafruit IO

Adafruit IO se eligió como la plataforma en la nube más adecuada para la visualización de los datos recopilados por el nodo de sensado. Esta plataforma permite que microcontroladores y sistemas con conexión a internet puedan enviar y visualizar datos de forma sencilla, intuitiva y eficiente. A través de su interfaz se pueden crear tableros o dashboards con bloques gráficos, indicadores, interruptores y medidores que muestran en tiempo real la información enviada, con un límite de 30 instancias de datos por minuto.

El funcionamiento interno de Adafruit IO se estructura alrededor del concepto de fuentes o “feeds”, que son canales de datos donde cada variable medida por el sistema es enviada. Cada vez que un dispositivo actualiza el valor de una variable en un feed, la plataforma almacena el dato y lo distribuye a los tableros que lo utilizan, permitiendo sincronización casi instantánea entre el sistema y la nube, se puede observar el dashboard utilizado en la Figura 10 El protocolo MQTT juega un papel fundamental en esta arquitectura, ya que Adafruit IO lo utiliza como su medio principal de comunicación. Esta estructura junto con este protocolo reduce la carga de red, mejora la eficiencia energética y garantiza una comunicación confiable incluso con conexiones inestables o de baja velocidad. Gracias a esto, Adafruit IO puede manejar de manera ligera y escalable la transmisión continua de datos entre dispositivos y la nube, convirtiéndose en una herramienta ideal para este proyecto.

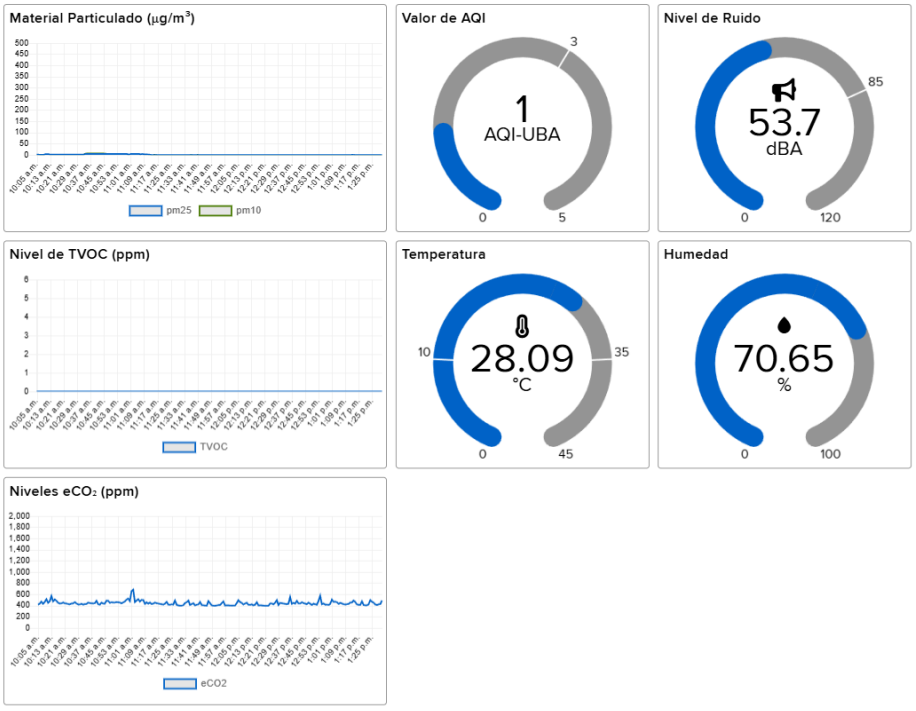

Figura 10: Dashboard de datos en la plataforma Adafruit IO.

9.2.3. Altium

Para el diseño de la placa de circuito impreso (PCB) del nodo de sensado se utilizó el software Altium Designer, una de las herramientas más completas y profesionales disponibles para el desarrollo de sistemas electrónicos. Este programa permite realizar de forma integrada la captura esquemática, el diseño y ruteo de la PCB, la verificación de reglas eléctricas y la generación de archivos de fabricación, todo dentro de un mismo entorno. Además, ofrece visualización tridimensional de la placa y los componentes, lo que facilita la detección de errores y la validación mecánica del diseño antes de su producción.

En esta plataforma se diseñó una placa de 52 mm x 79 mm, observable en la Figura 11 que posee dos capas conductivas donde se realizó el trazado de las líneas de transmisión. Para estas se utilizó un tamaño de 1 mm para las trazas de alimentación y 0.5 mm para las trazas de datos, con el fin de mantener una buena capacidad de manejo de corriente para cada dispositivo y evitar interferencias por altas frecuencias en las líneas de transmisión de datos. Seguidamente se puede observar un diagrama del circuito impreso diseñado con la capa superior en rojo y la capa inferior en celeste.

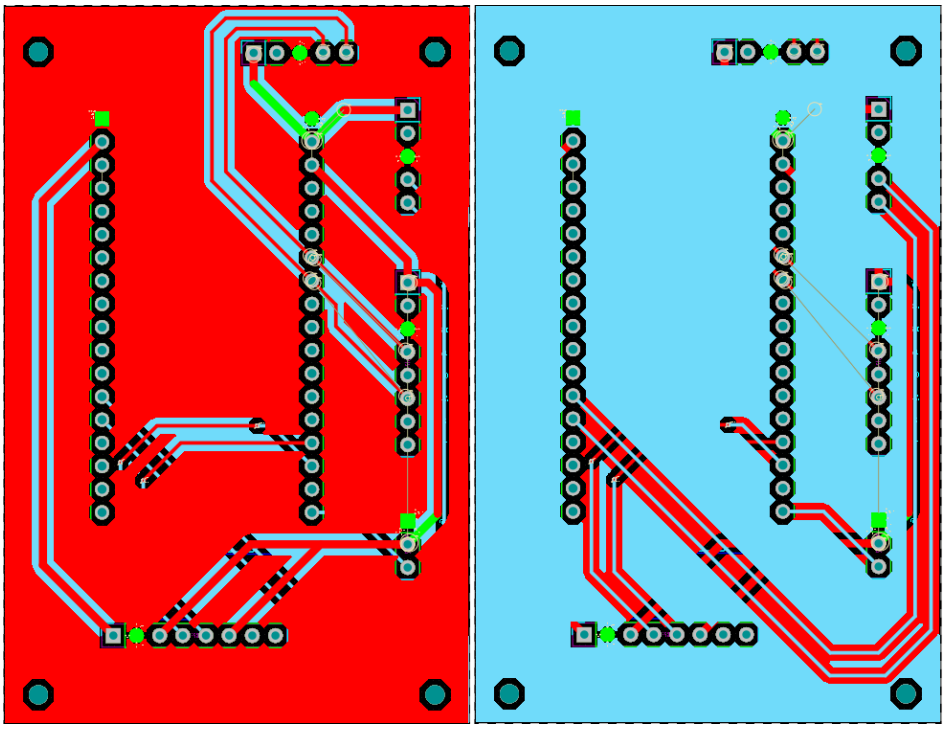

Figura 11: Diagrama de PCB en Altium Designer.

9.3. Diagramas de Bloques

A continuación se presenta un diagrama de bloques básico para ilustrar la solución diseñada en este proyecto.

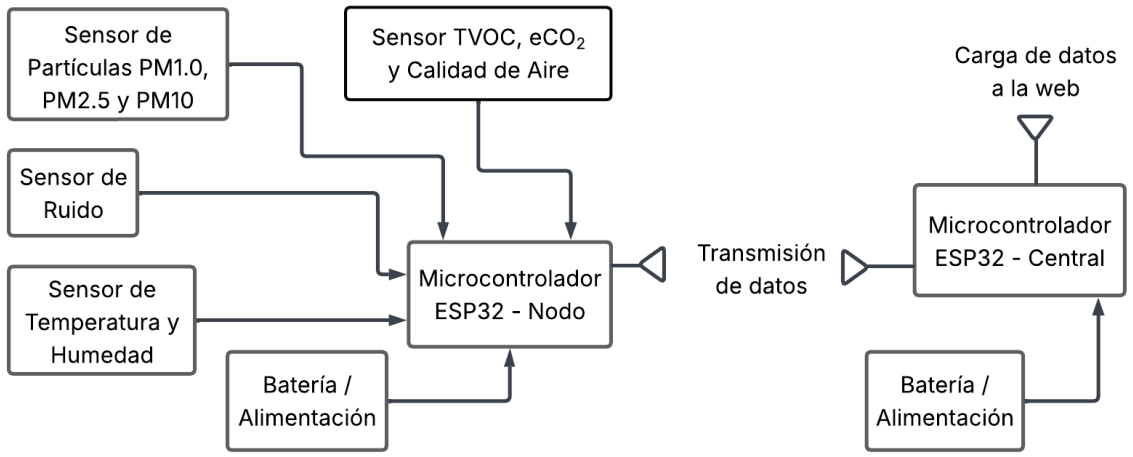

Figura 12: Diagrama de bloques ilustrativo de la solución seleccionada.

El sistema físico final propuesto presenta una fácil integración gracias a la versatilidad y simplicidad de solo requerir un puerto UART, dos puertos I^2C y un puerto analógico para comunicación con el microcontrolador. Seguidamente se presenta un diagrama detallando las conexiones entre los componentes del sistema.

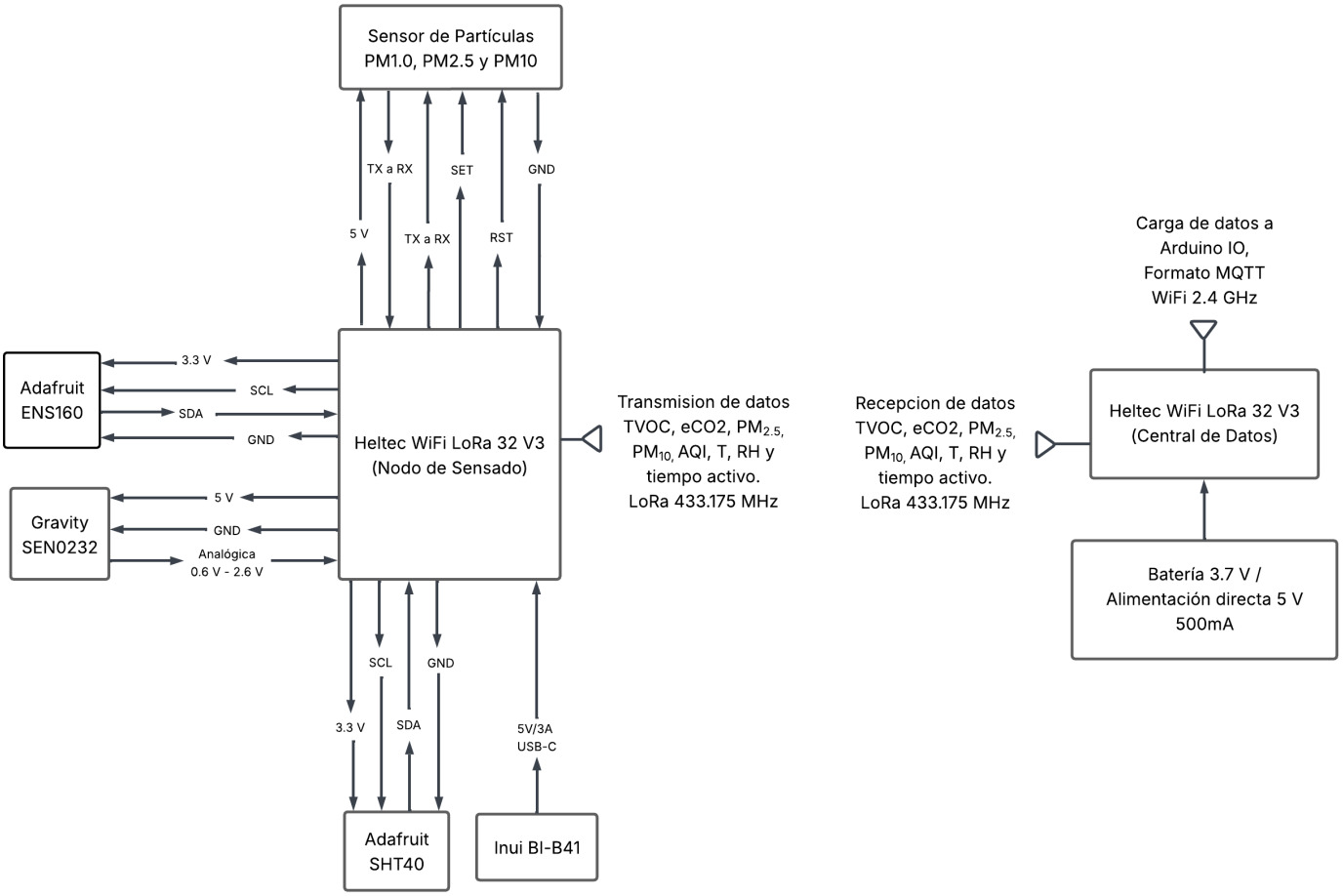

Figura 13: Diagrama de bloques avanzado.

10. Procedimiento de desarrollo

Para iniciar se realizó un repaso de programación en C++. Este lenguaje es necesario para programar el microcontrolador. También se realizó un repaso del funcionamiento de la plataforma de programación a utilizar, Arduino IDE. Esto se cumple con ayuda de tutoriales, guías y cadenas de mensajes en foros varios.

Seguidamente se realiza la integración de los sensores con el microcontrolador y se verifica que este reciba datos ambientales; para este cometido se buscaron las hojas de datos de cada componente con el fin de realizar las conexiones de manera correcta. Para cada sensor se escribió un código de prueba, cuyo objetivo fue lograr obtener datos de estos individualmente. Este enfoque permitió comprender y comprobar el correcto funcionamiento de cada uno de los sensores. Se realizaron simples pruebas de funcionamiento y se compararon los datos obtenidos con los niveles definidos en el estándar de calidad de aire AQI-UBA y los reglamentos de contaminación de ruido de Costa Rica.

Con los sensores funcionando correctamente se procedió a escribir un programa modular para integrar todos estos componentes bajo un solo código. Para esto se crea un programa principal (Main) para manejar el flujo del código y, para los sensores, se crea un programa de encabezado (header o .h) para declarar variables, clases y funciones, y un programa de inicialización (.cpp) para definir la lógica de cada función. Una vez programado, se deben realizar pruebas para verificar el correcto funcionamiento de los sensores del sistema en conjunto. Los sensores se conectaron al microcontrolador utilizando una protoboard y jumpers, se verificó la continuidad en cada pin, se carga el programa modular al ESP32 y se realiza una prueba de funcionamiento comparando, de igual manera, con los diferentes estándares definidos.

Luego de comprobar el correcto funcionamiento de los sensores se debe programar la funcionalidad de comunicación inalámbrica por LoRa a 433.175 MHz entre el microcontrolador del nodo de sensado y la central de recolección de datos. Para verificar la certeza de los datos recibidos se agregaron medidas de Relación Señal-Ruido (Signal to Noise Ratio o SNR) e Indicador de Intensidad de Señal Recibida (Received Signal Strength Indicator o RSSI) para las comunicaciones entre ambos microcontroladores. De esta manera se puede tener una referencia para asegurar que la pérdida de datos es mínima o lo más baja posible para no comprometer la integridad de la información que se desea recibir. Una vez comprobada la recepción de datos íntegra, en caso de ser necesario, se debe programar la central de recolección para que realice la normalización de los datos recibidos y que estos mantengan consistencia, facilitando su interpretación por los usuarios.

Luego de esto, se programa la funcionalidad de carga a la nube. Para esto, se crea una cuenta en Adafruit IO y se obtiene una "llave" que permite la carga de datos al dash-

board diseñado. Los datos son agrupados por variable y son cargados a la plataforma en la nube utilizando la capacidad WiFi del ESP32. Existen diferentes estándares de formato de datos para este objetivo; en el caso de esta aplicación se escogió el protocolo MQTT, ya que este es el más apto gracias a su compatibilidad con la plataforma. Para finalizar, se verifica la carga de datos en la nube al recibir datos y visualizar el cambio en cada bloque gráfico.

11. Montaje del sistema

Para el montaje del sistema de sensado propuesto se utilizó primero una protoboard para realizar las pruebas de funcionamiento de los componentes antes de ser montados en el circuito impreso para el nodo, observable en la Figura 14. Una vez que se decidió el orden de los componentes se procedió a diseñar la placa para montar el sistema; para esto se utilizó el programa Altium Designer. En este software se diseñó el diagrama inicial del sistema, se importó a la herramienta de diseño y se conectaron los componentes al microcontrolador del nodo con líneas de 1 mm para poder y tierra y de 0.5 mm para las líneas de transmisión de datos, ver Figura 11. Se agregó un puerto I^2C extra para posible escalamiento futuro. Una vez conectado todo, se importaron los archivos Gerber para poder enviar la placa a ser creada. Para esto se optó por ordenarla a la compañía PCBWay y, como respaldo, solicitar su fabricación en la Escuela de Ingeniería Electrónica del Tecnológico de Costa Rica.

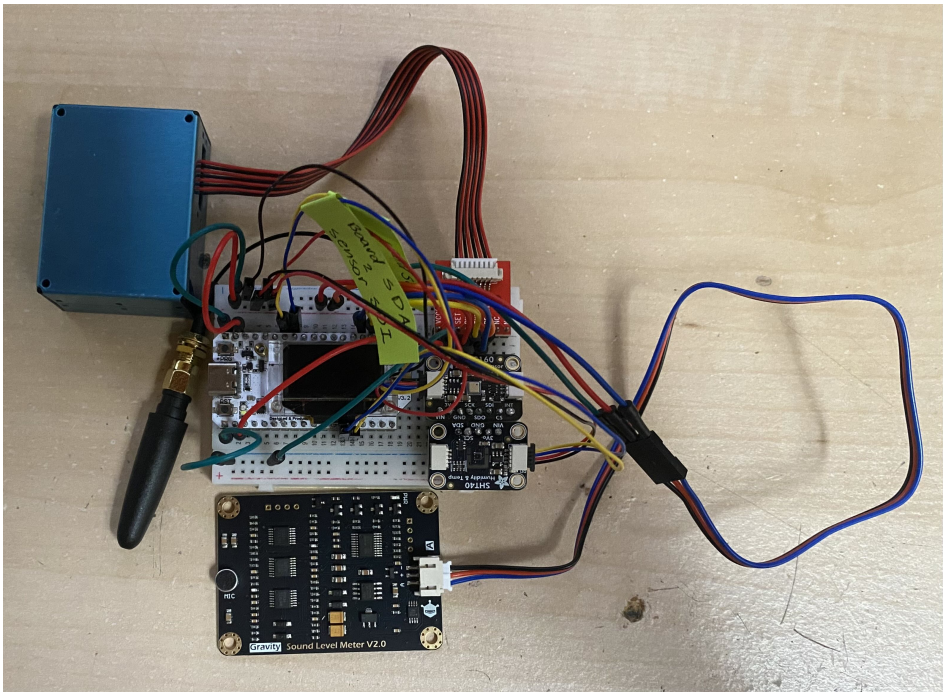

Figura 14: Componentes conectados a una protoboard.

Luego de obtener la placa diseñada, se soldaron los componentes en sus respectivos lugares, observable en la Figura 15.

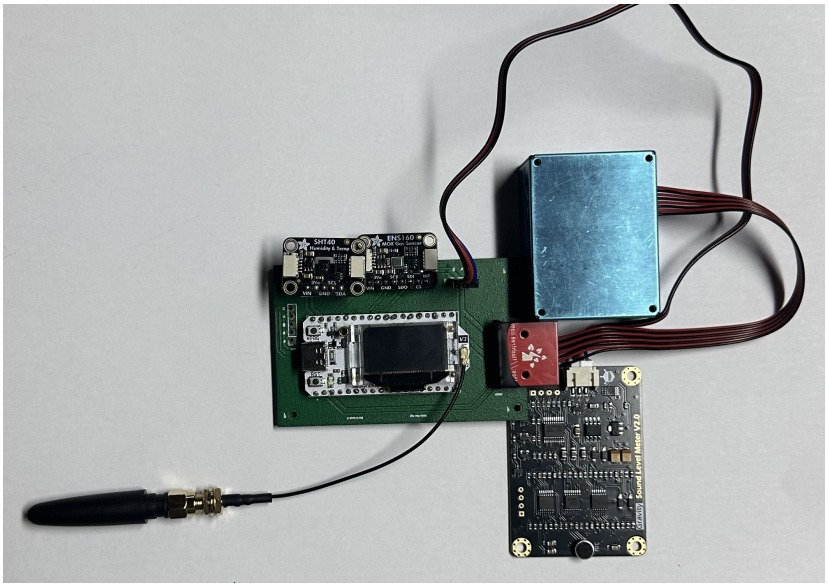

Figura 15: Nodo de sensado integrado en el PCB.

Esta placa fue diseñada con agujeros para poder ser montada en el encapsulado del nodo; de esta manera el PCB queda ajustado, evitando el peligro de daños por movimientos bruscos, según se muestra en la Figura 16. Finalmente se cierra el encapsulado y el nodo de sensado queda asegurado dentro de éste y se coloca en el lugar donde se desea realizar el monitoreo.

Figura 16: Sistema montado en el PCB dentro del encapsulado.

En la Tabla 16 se pueden observar las conexiones realizadas entre los sensores del nodo y el microcontrolador.

Componente	**Conexión**	**Pin en el microcontrolador**
Adafruit ENS160	Pin 1 - Vcc	Header J3 Pin 3 - 3v3
	Pin 3 - GND	Header J3 Pin 1 - GND
	Pin 4 - SCL	Header J3 Pin 7 - GPIO42
	Pin 6 - SDA	Header J3 Pin 8 - GPIO41
Adafruit SHT40	Pin 1 - Vcc	Header J3 Pin 2 - 3v3
	Pin 3 - GND	Header J3 Pin 1 - GND
	Pin 4 - SCL	Header J2 Pin 14 - GPIO48
	Pin 5 - SDA	Header J2 Pin 13 - GPIO47
PMS5003	Pin 1 - Vcc	Header J2 Pin 2 - 5 V
	Pin 2 - GND	Header J2 Pin 1 - GND
	Pin 3 - SET	Header J3 Pin 3 - 3v3
	Pin 4 - RX	Header J3 Pin 16 - GPIO5
	Pin 5 - TX	Header J3 Pin 15 - GPIO4
	Pin 6 - RST	Header J3 Pin 3 - 3v3
Gravity SEN0232	Pin 1 - GND	Header J3 Pin 1 - GND
	Pin 2 - Vcc	Header J3 Pin 3 - 3v3
	Pin 3 - OUT	Header J3 Pin 18 - GPIO 7

Tabla 16: Conexiones realizadas entre los sensores y el microcontrolador.

12. Programación del sistema

En esta sección se explica el código para cada módulo del nodo de sensado y de la central de datos. Inicialmente se debieron buscar e instalar todas las librerías necesarias para el desarrollo del código del sistema de sensado, observables en la Figura 17, algunas de estas son provistas por los vendedores de los módulos de sensado elegidos; para otras se debió realizar una investigación previa antes de su instalación para verificar compatibilidad con los componentes seleccionados y así evitar errores.

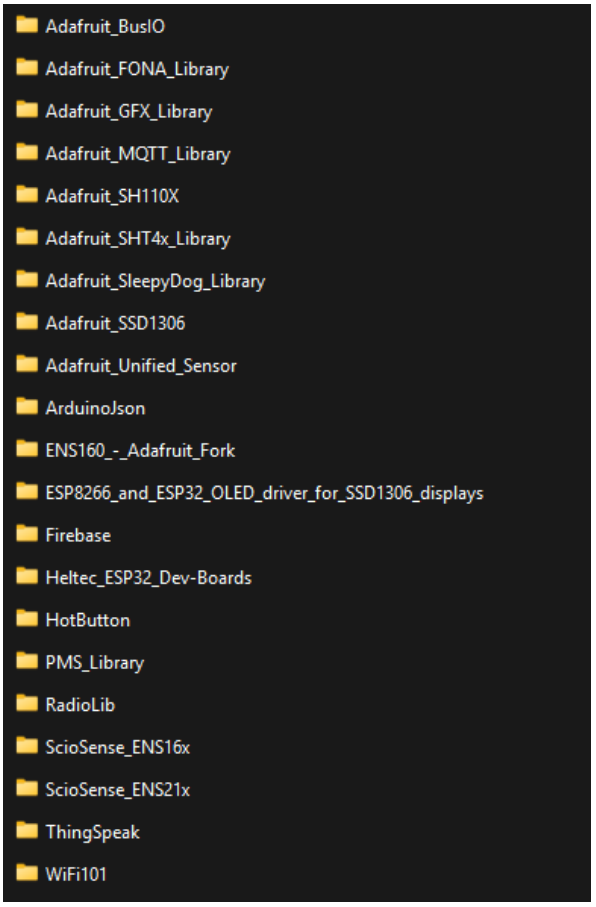

Figura 17: Librerías utilizadas con Arduino IDE.

Es importante establecer el ciclo de funcionamiento del sistema. Al inicializar ambos, el nodo de sensado y la central de datos, suceden dos procesos. El nodo inicia la conexión con los sensores y verifica la comunicación con cada uno de estos, observable en la Figura 18. Luego de este proceso, el nodo espera por la captura de datos, la cual sucede cada 3 segundos, esto debido a el tiempo de respuesta del sensor ENS160, los

demás sensores pueden tomar datos de manera casi instantánea pero se considera que 20 muestras por minuto es una buena base para verificar el funcionamiento correcto del nodo de sensado.

Estos datos se guardan y se repite el ciclo de captura hasta que, luego de un minuto, se calcula el promedio de los valores guardados para cada variable y se transmite por radiofrecuencia a la central de datos, y se da confirmación del paquete enviado, este último siendo de 23 bytes y observable en la Figura 19.

```
ESP-ROM:esp32s3-20210327
Build:Mar 27 2021
rst:0x1 (POWERON),boot:0x29 (SPI_FAST_FLASH_BOOT)
SPIWP:0xee
mode:DIO, clock div:1
load:0x3fce2820,len:0x116c
load:0x403c8700,len:0xc2c
load:0x403cb700,len:0x3108
entry 0x403c88b8

Scanning Wire...
  - I2C device at 0x44

Scanning Wire1...
  - I2C device at 0x53
ENS160 ready at 0x53
```

Figura 18: Activación del nodo de sensado y verificación de conexión con sensores.

```
-> TX seq=0 bytes=23 ok
```

Figura 19: Confirmación de envío de paquete de datos por LoRa.

La central de datos se inicializa de manera similar al nodo, con la diferencia de que este realiza una verificación de la conexión a WiFi y la plataforma de Adafruit IO. Una vez dado este proceso, se activa la antena y se espera por paquetes, visible en la Figura 20. Al recibir un paquete de datos, observable en la Figura 21 y la Figura 22, se normaliza el valor de TVOC de ppb a ppm para que esta medida esté estandarizada junto con eCO_2, y se suben los datos ambientales recibidos a la plataforma en la nube, donde estos pueden ser visualizados por el usuario, observable en la Figura 10.

```
ESP-ROM:esp32s3-20210327
Build:Mar 27 2021
rst:0x1 (POWERON),boot:0x8 (SPI_FAST_FLASH_BOOT)
SPIWP:0xee
mode:DIO, clock div:1
load:0x3fce2820,len:0x116c
load:0x403c8700,len:0xc2c
load:0x403cb700,len:0x3108
entry 0x403c88b8
LoRa receiver ready @433.175 MHz
[WiFi] Connected: 192.168.100.84 RSSI=-86
Receiver boot complete
Awaiting data...
```

Figura 20: Activación de la central de datos.

```
[RX] seq=0 node=1 RSSI=-75 dBm SNR=10.8 dB
   T: 27.29 °C | RH: 84.63 % | TVOC: 0.031 ppm | eCO2: 419 | AQI: 1 | PM2.5: 2 | PM10: 2 | dBA: 56.0
Awaiting data...
```

Figura 21: Confirmación de recepción de paquete de datos por LoRa.

Figura 22: Central confirma la recepción de datos.

El proceso de funcionamiento del sistema se puede visualizar en el diagrama de flujo de la Figura 23.

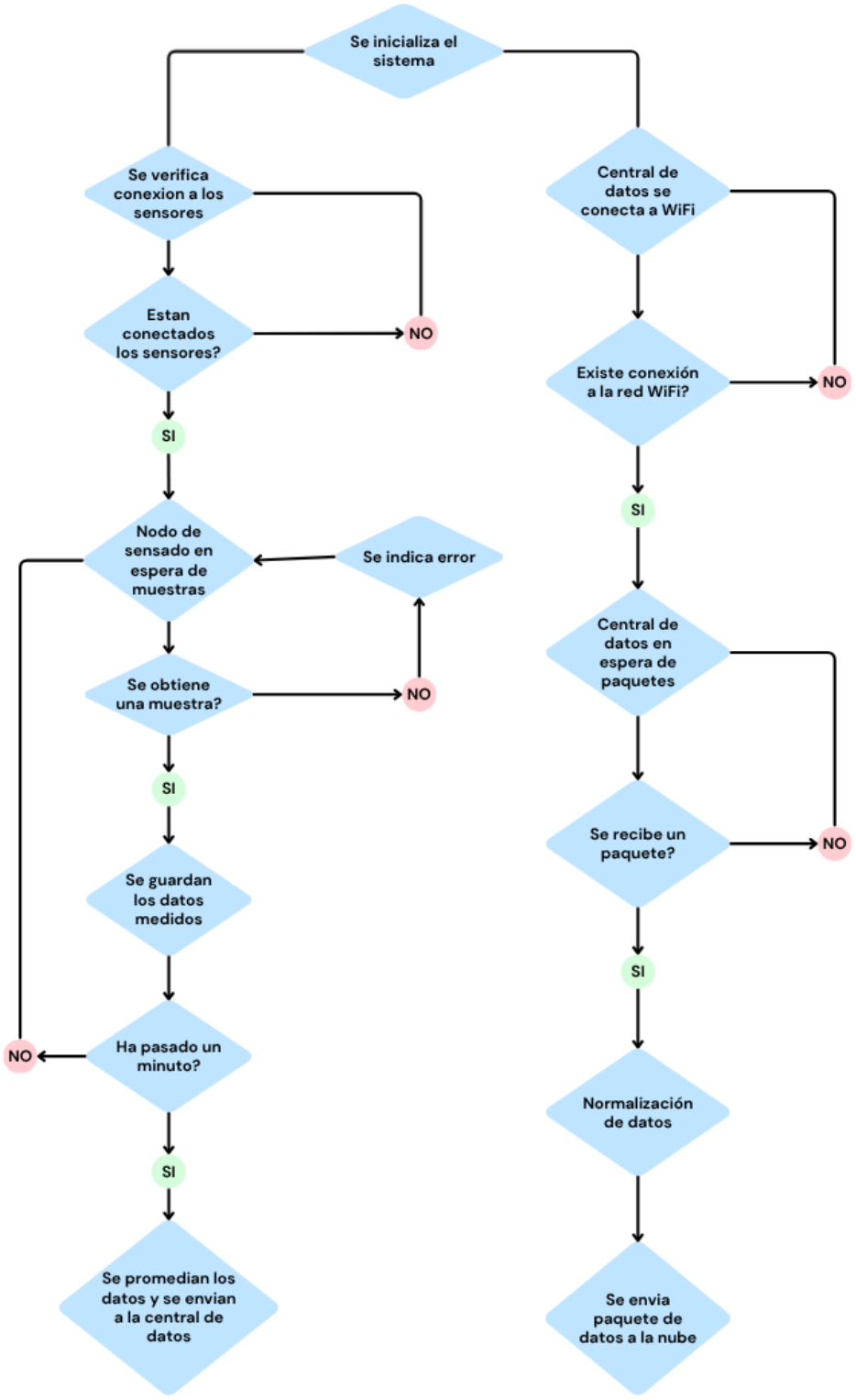

Figura 23: Diagrama de flujo del sistema de sensado.

Seguidamente se presenta una explicación para cada sección del código implementado en el desarrollo del sistema, en la sección 19 se presenta el código completo.

12.1. Nodo de Sensado

12.1.1. Main

El programa inicializa dos buses I^2C (Wire y Wire1) y configura cuatro módulos: SHT40, ENS160, PMS5003 y SEN0232; además, también inicializa el transmisor LoRa. Se codifica un bucle para realizar un sondeo rápido del PMS5003 para no perder tomas o poder, causando que el sensor se apague, y una impresión periódica donde se toma una captura. Se lee el SHT40 y se aplica compensación; se lee el ENS160 y, finalmente, se lee también el SEN0232.

12.1.2. Sensor Adafruit SHT40

Esta sección cubre la programación de los bloques relacionados con el sensor Adafruit SHT40, para esto se instala la librería *Adafruit _SHT4x*.

El header funciona como una interfaz para el SHT40 usando la librería de Adafruit. Se importa el driver, se declara la clase SHT40Sensor con una API mínima, se inicializa el bus I^2C y se mantiene el estado interno de cada variable (temperatura y humedad relativa). Su función es entregar al programa principal una API limpia, estable y fácil de utilizar.

Seguidamente, se inicializa el sensor, se conecta con el bus indicado (por defecto Wire) y se configuran parámetros de alta precisión (mejor SNR) y el calentador o *heater* apagado, lo cual es importante para poder realizar la compensación en otros sensores.

Finalmente, se realiza una medición y al finalizar se copian los valores a las variables internas y a las referencias de salida, lo que permite obtener nuevos datos sin choques.

12.1.3. Sensor Adafruit ENS160

Para la programación de este sensor se debió instalar la librería *ScioSense _ENS16x*.

El header hace una declaración de clase y se guarda un puntero al objeto ENS160. Se inicializa el driver y el bus I^2C y se utilizan *üpdate"* y *"hasNewData"* para obtener y actualizar datos. Se usan comandos de adquisición de variables (AQI/TVOC/eCO_2) y *setCompensation* escribe datos de temperatura y humedad relativa del sensor SHT40 a registros dedicados del ENS160.

Se incluyen las librerías del sensor y el header, se inicializa el driver y se realiza una consulta para verificar si hay nuevos datos y se actualizan los registros. Seguidamente,

se inicializa la obtención de medidas de AQI, TVOC y eCO_2.

Se escribe la compensación ambiental (T en Kelvin × 64 y %RH × 512) según la hoja de datos del sensor [42] y se guardan los datos obtenidos directamente en los registros del ENS160 desde el bus I^2C con un orden de LSB a MSB.

12.1.4. Sensor Plantower PMS5003

Para la programación de este sensor se requirió la instalación de la librería *PMS.h*.

El header declara una clase para el sensor y se asignan los pines en el ESP32 y pines opcionales de SET y RESET activos en bajo. Se devuelve true cuando se decodifica un dato válido, se actualiza la memoria y se devuelve el último valor medido.

Se inicializa el sensor y se configura el puerto UART a 9600 baudios para la comunicación con el microcontrolador, se ponen SET y RESET en estado inactivo (inactivos en HIGH, activos en LOW) y se usa la librería para parsear datos válidos. En cada lectura de datos exitosa se mide material particulado PM1,0, PM2,5 y PM_{10} en ($\mu g/m^3$).

12.1.5. Sensor Gravity SEN0232

El header define una clase para usar el sensor de ruido SEN0232. La clase guarda el pin del ADC, realiza una lectura cuando se llama a read(), convierte esa lectura a milivoltios y estima el nivel de ruido en dB(A). Además, ofrece métodos para consultar el último valor calculado sin volver a medir (dBA() y mV()). Con este encapsulado, el programa principal queda limpio y es fácil cambiar o re-calibrar el sensor sin tocar el resto del código.

Se implementa e inicializa la lectura de datos del sensor SEN0232, primero se define una rutina interna que configura el ADC del ESP32, toma 32 muestras para promediar y convierte el resultado a milivoltios con 3.3 V de referencia. Luego aplica una calibración lineal que usa un punto de "cero" eléctrico y una pendiente para transformar el voltaje a dBA, y devuelve el resultado dentro de un rango de 30–120 dBA. Se registra el pin ADC y se ejecuta el flujo completo de leer mV, convertir a dBA, actualizar la caché interna y devolver ambos valores. De esta forma, el sistema obtiene mediciones de nivel sonoro estables y fáciles de interpretar.

12.1.6. Transmisión de datos por LoRa

El header define una clase para transmitir por LoRa usando RadioLib; se utiliza begin() para inicializar el radio y transmit() para enviar un paquete de datos o "payload". Internamente se guarda un puntero a una estructura privada que contiene los objetos de RadioLib (SPI, módulo y radio), manteniendo el header limpio y sin dependencias.

Se inicializa fijando los pines del módulo LoRa (SX1262), la configuración RF de los parámetros: frecuencia, ancho de banda, factor de dispersión (spread factor o SF), tasa de codificación (coding rate o CR), preámbulo, palabra de sincronización y potencia de transmisión. Luego se crea la estructura Impl con SPIClass, Module y SX1262. En begin() se inicia el SPI, se configura el radio a la frecuencia indicada y se aplican los parámetros establecidos; la llamada a Dio2 es importante en esta versión del microcontrolador para manejar el switch de radiofrecuencia interno. En transmit() se envía el payload como bloque binario y se imprime una confirmación breve de la transmisión.

Finalmente se define un archivo para establecer el paquete de datos que se desea enviar a la central de datos. Este "payload.[es] una estructura compacta de 23 bytes que contiene los datos del nodo. Los datos enviados son temperatura, humedad, ruido, TVOC, eCO_2, calidad de aire según el estándar AQI-UBA, material particulado, número de secuencia y tiempo que lleva encendido el nodo en segundos.

12.2. Central de Datos

12.2.1. Main

El programa principal para la central de datos se encarga de varias tareas: enciende y prepara la pantalla OLED integrada, muestra una confirmación de arranque y un mensaje de espera por datos, e incorpora una línea de estado a la red WiFi seleccionada en la pantalla para verificar que hay conectividad. En paralelo inicializa el receptor LoRa para una frecuencia de 433.175 MHz y los clientes de red WiFiLink y AIOClient para publicar los datos en la nube. En el lazo principal escucha por paquetes enviados por LoRa y, cuando recibe uno, registra la telemetría recibida y publica los datos en el servidor de Adafruit IO. La pantalla OLED cambia a una vista de "RX OK" con la secuencia de datos, RSSI y SNR; si no llegan más paquetes en 10 segundos, vuelve al estado de espera. Así, el nodo receptor ofrece visibilidad local (OLED y serial), conectividad de backend y escalamiento de datos al normalizar TVOC de ppb a ppm.

12.2.2. Recepción de datos por LoRa

El header inicializa el chip SX1262 y escucha por paquetes; al recibirlos entrega los datos del payload y reporta las métricas de comunicación RSSI y SNR, junto con el tiempo activo del nodo de sensado. El detalle importante en este encabezado es que toda la complejidad de RadioLib se maneja dentro de un puntero, lo que mantiene el header limpio y evita dependencias innecesarias en otros módulos del programa, y facilita cambios futuros en la implementación. El resto del sistema solo ve un receptor que arranca y entrega paquetes decodificados con las métricas de radiofrecuencia.

Se inicializa el SX1262 con RadioLib, el bus SPI y el módulo de radio, y se pone el

receptor en escucha continua por paquetes a una frecuencia de 433.175 MHz. Se activa Dio2 para las comunicaciones por radiofrecuencia y se ajustan los parámetros de enlace para que estos sean iguales a los del transmisor. Se marca una bandera cuando llega un paquete; si esta está activa, se lee el bloque binario de datos recibidos, se copia a la salida y se capturan las métricas de RSSI y SNR antes de volver automáticamente a esperar la recepción de un nuevo paquete.

Para la recepción de datos es necesario añadir una copia del archivo payload al nodo receptor para que este sepa qué información esperar recibir del nodo de sensado.

12.2.3. Conexión a WiFi

El header encapsula la funcionalidad de conexión inalámbrica, inicia la conexión y devuelve true si el dispositivo está conectado a una red WiFi. Este header permite que otras partes del programa usen la clase WiFiLink sin preocuparse por los detalles internos de cómo se realiza la conexión a la red.

Se inicializa la lógica de conexión WiFi y se configura el módulo en modo estación (STA o Station Mode), lo que significa que el dispositivo actuará como cliente para poder conectarse a una red existente. Utiliza las credenciales definidas en el archivo de credenciales para iniciar la conexión.

El código espera 8 segundos para establecer la conexión; si se da una conexión exitosa se muestra la dirección IP local y la intensidad de la señal recibida o RSSI. Si no se conecta en ese tiempo, se muestra un mensaje de error indicando la falla de conexión. Se agrega un bloque que permite verificar si el dispositivo sigue conectado a la red, esto con el objetivo de reconectarse en caso necesario o manejar errores de comunicación.

Para ambas la conexión por WiFi y la carga de datos a la nube se requiere el uso de credenciales para acceder a las mismas, por lo que se agregó un módulo para contener esta información y tenerla centralizada y a mano para su uso con las funcionalidades de WiFi y carga a Adafruit IO.

12.2.4. Carga de datos a Arduino IO

Esta porción del código para la central de datos se encarga de enviar los datos ambientales a la plataforma en la nube Adafruit IO usando el protocolo MQTT.

Este header encapsula la lógica necesaria para conectarse a Adafruit IO y publicar datos para ser visualizados mediante la interfaz gráfica. Cada comando pubX() representa alguno de los datos ambientales recibidos por el nodo de sensado; este se envía a su *"feed"*, que es el canal donde se publica el dato recibido por el nodo. Mediante el uso de un ciclo se realiza un llamado periódico para mantener la conexión activa.

Este archivo implementa la lógica de conexión y publicación de datos a Adafruit IO utilizando el protocolo MQTT. Se configura el cliente MQTT con el servidor de Adafruit IO y las credenciales de conexión a la nube. Se realiza una verificación de conexión para reconectar el dispositivo si la conexión con el servidor se pierde, si existe un enlace entonces se publica el valor del dato enviado a su feed específico.

13. Pruebas de funcionamiento y análisis de resultados

En esta sección se presentan los procedimientos de las pruebas de funcionamiento realizadas para el sistema, los resultados obtenidos y el análisis de estos basándose en los estándares definidos en los fundamentos teóricos y en las métricas de calidad e integridad de la señal. Se van a realizar una prueba de autonomía y una prueba de captura de datos en un ambiente urbano abierto a tráfico de personas y vehículos.

13.1. Funcionamiento del sistema en un ambiente urbano

13.1.1. Instalación del Sistema

Para esta prueba se colocó el sistema en un área abierta que presenta un flujo moderado de tráfico vehicular y de personas. Se instaló en el exterior del restaurante *The House*, observable en la Figura 24, al costado oeste de los Tribunales de Justicia de San Ramón de Alajuela. Esta ubicación se encuentra a una distancia aproximada de 100 metros de donde está conectada y en espera la central de datos.

Figura 24: Posicionamiento del nodo de sensado en un ambiente urbano.

La central de datos se conecta a una computadora con Arduino IDE instalado para el monitoreo de datos y métricas recibidas utilizando el monitor serial a 115200 baudios. Se inicia con un spread factor SF7 preliminarmente para verificar la integridad de la señal. Al recibir los primeros 5 paquetes se obtienen los resultados de SNR y RSSI observables en la Tabla 17.

RSSI (dBm)	SNR (dB)
-75	10.5
-77	10.5
-77	10.8
-78	10.3
-77	11.8

Tabla 17: Valores de RSSI y SNR con un factor de dispersión SF7.

Considerando que para una comunicación LoRa efectiva se requiere un RSSI entre –30 dBm y –70 dBm, se decide ajustar el valor de factor de dispersión a SF9 y se vuelve a encender el sistema para verificar la integridad de la señal. Los primeros 5 paquetes indican nuevas métricas de SNR y RSSI observables en la Tabla 18.

RSSI (dBm)	SNR (dB)
-55	10.8
-57	10.3
-57	10.5
-58	11.3
-57	10.5

Tabla 18: Valores de RSSI y SNR con un factor de dispersión SF9.

13.1.2. Captura y carga de datos ambientales urbanos a la nube

Una vez verificado un buen enlace, se realizan mediciones de datos por un periodo de aproximadamente 5 horas y 25 minutos, de 10:04 a.m. a 3:29 p.m. No se realiza una prueba de mayor duración debido a factores adversos como mal clima y peligro de hurto, lo cual pondría en riesgo al nodo de sensado. Durante este periodo de 325 minutos se espera recibir 325 métricas de enlace y paquetes de datos ambientales promediados por parte del nodo de sensado, y finalmente estos se envían a la plataforma Adafruit IO donde pueden ser visualizados.

En las siguientes figuras se pueden observar gráficas de los resultados obtenidos para las métricas de SNR, RSSI durante este periodo.

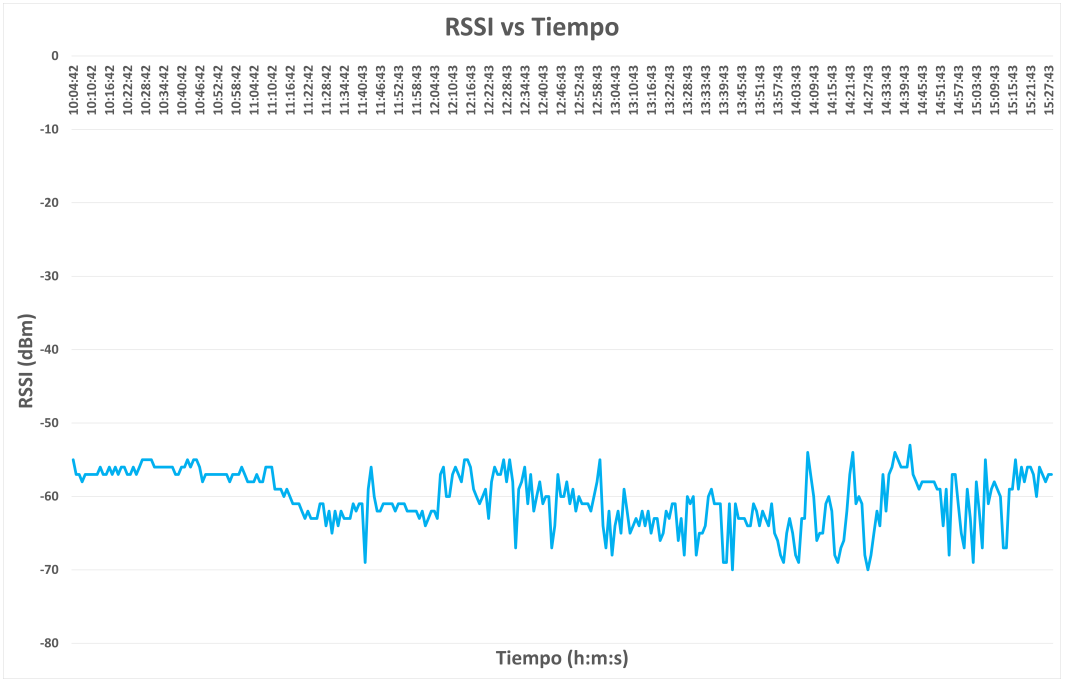

Figura 25: Gráfica de comportamiento de RSSI durante el periodo de prueba.

Al recibir los primeros paquetes con un factor de dispersión SF7 se observa un SNR alto rondando los 11 dB, indicando que se tiene un buen enlace y buena calidad en la señal; aun así, también se observa un RSSI que se encuentra por debajo de los -70 dBm, lo que permite intuir que, aunque la señal recibida sea clara y distinguible del piso de ruido, esta posee una potencia baja, por lo que puede ser inestable con posible pérdida de datos. Al cambiar el factor de dispersión a SF9 se puede notar un considerable aumento en el RSSI de la transmisión en la Figura 25, indicando un aumento en la potencia de la señal recibida dentro del umbral de -50 dBm a -70 dBm, donde ya se puede considerar que la transmisión de datos es buena.

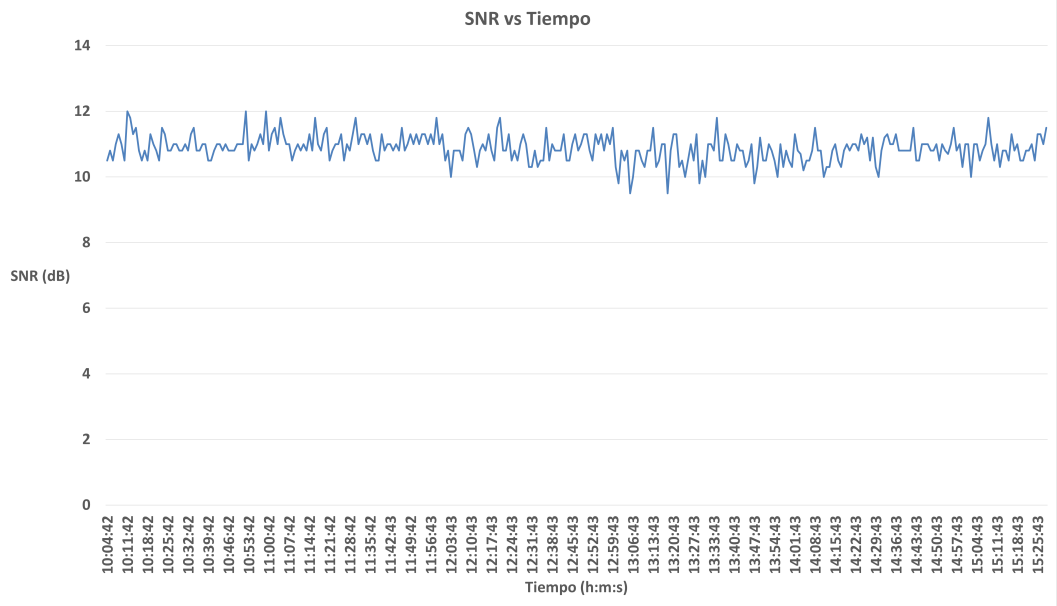

Figura 26: Gráfica de comportamiento de SNR durante el periodo de prueba.

Se puede observar en la gráfica del SNR de la Figura 26 un comportamiento constante entre 9 dB y 12 dB, indicando un enlace fuerte y confiable. Al comparar estos datos con los datos de RSSI es posible notar una degradación en la estabilidad de ambos parámetros: se observan variaciones más constantes en la segunda mitad del tiempo activo del nodo de sensado, pero se mantiene dentro del umbral aceptable. Estas variaciones pueden deberse a una combinación de factores como viento causando vibraciones que modifican el ángulo efectivo de radiación de la antena, reflexiones u otras interferencias, o cambios en las condiciones ambientales.

Es importante verificar que los datos que el usuario puede observar concuerden con los que la central de datos recibe, la plataforma Adafruit IO permite la descarga del historial completo de cada feed configurado, estos se comparan con los datos recibidos por la central de datos.

Para temperatura y humedad relativa se tienen las gráficas de las Figuras 27 y 28 respectivamente.

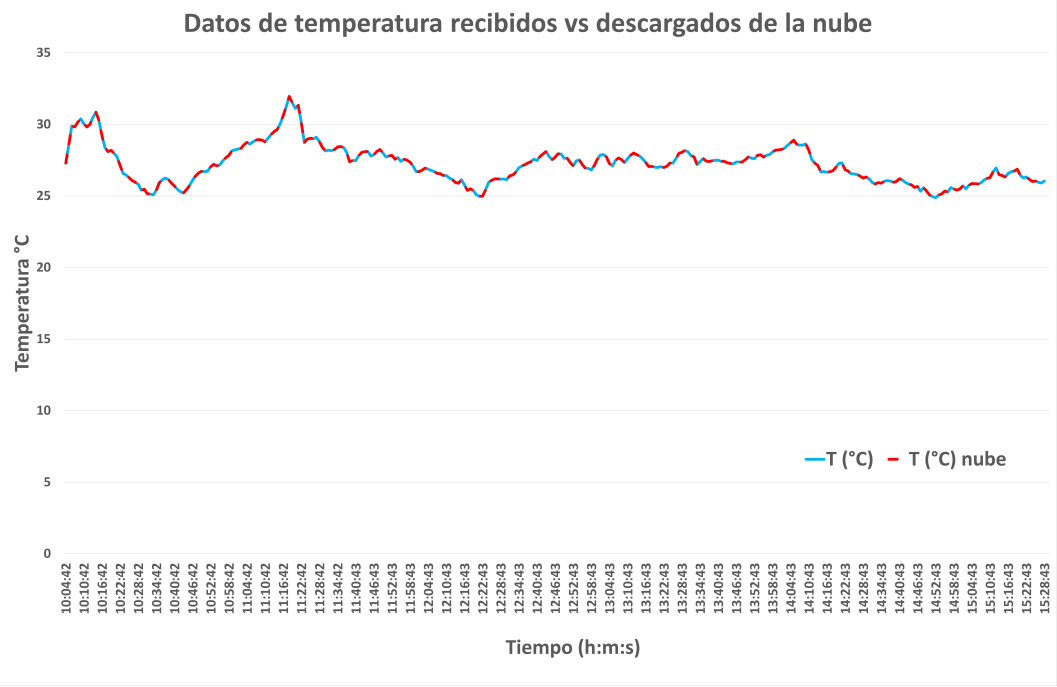

Figura 27: Gráfica de datos de temperatura recibidos vs descargados de la nube.

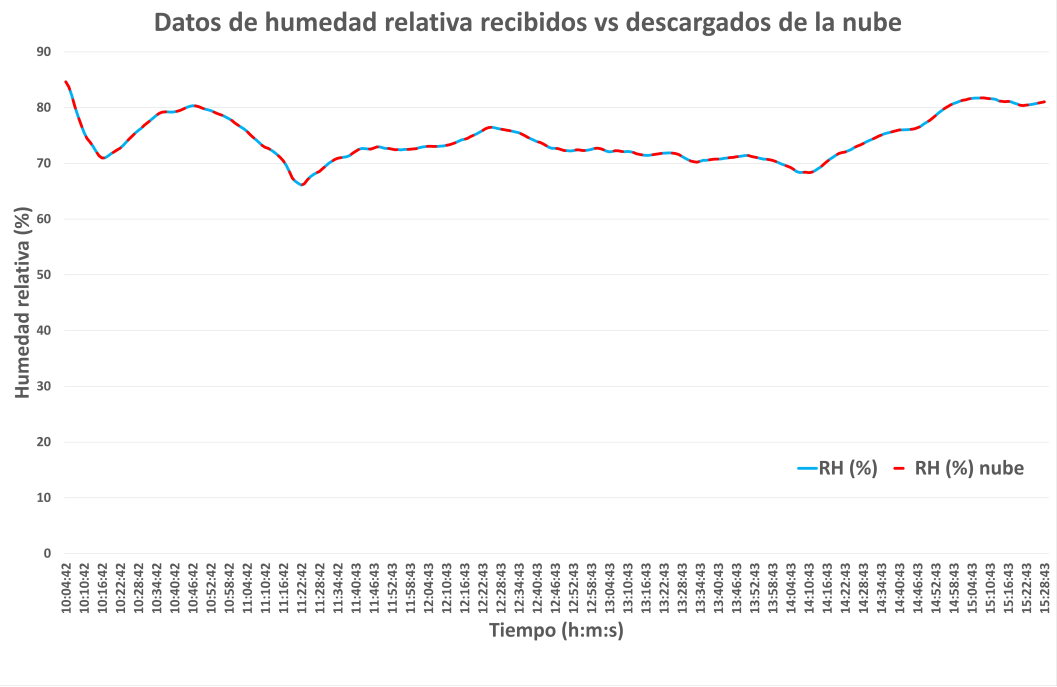

Figura 28: Gráfica de datos de humedad relativa recibidos vs descargados de la nube.

Al comparar las gráficas de temperatura y humedad relativa, se puede notar cómo, con los picos de temperatura, el porcentaje de humedad medida baja, en concordancia con periodos de luz solar directa. Al entrar a horas de la tarde, según la temporada climatológica de invierno y el hecho de que San Ramón es un cantón de altura, es normal que la acumulación de nubes bloquee la luz solar y se reduzca la temperatura. Estas mismas traen consigo niebla y lluvias, por lo que también aumenta la humedad; estos comportamientos se pueden observar en las gráficas de las Figuras 27 y 28.

Las medidas de TVOC, eCO_2 se observan en las gráficas de las Figuras 29 y 31 respectivamente.

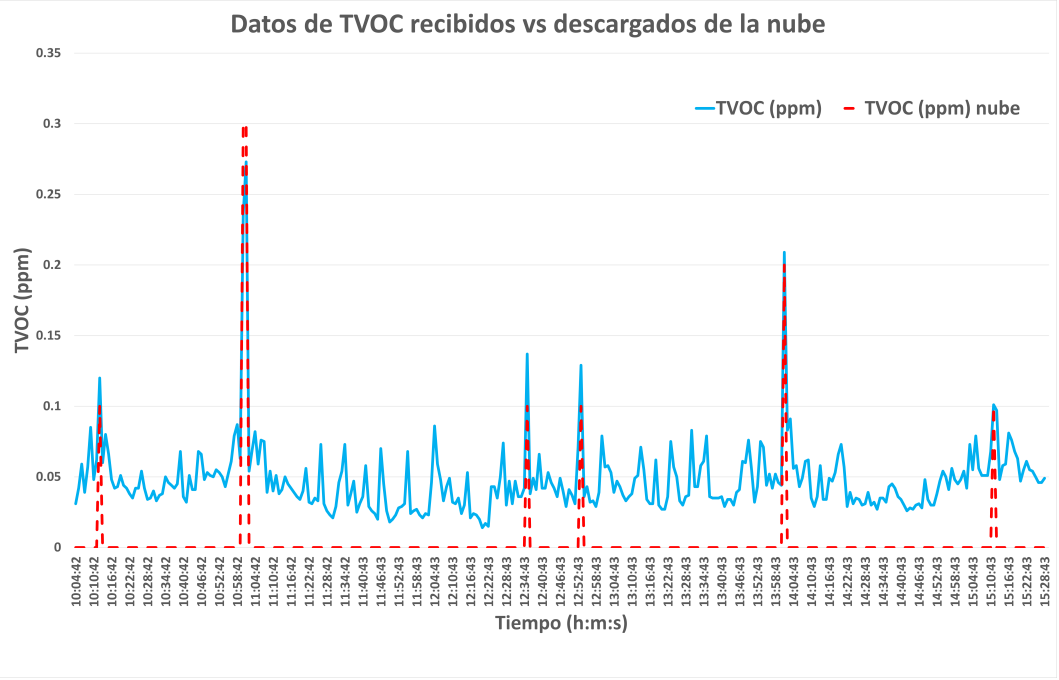

Figura 29: Gráfica de datos de TVOC recibidos vs descargados de la nube.

Observando la tabla de TVOC, lo más notable es cómo los datos descargados de la nube son 0 hasta que se da un pico que supera 0.1 ppm; esto es debido a la resolución limitada de la plataforma en la nube, que para ciertos tipos de variable tiene una resolución mínima de una cifra significativa. Estos picos en los niveles de TVOC pueden atribuirse a aumentos en la actividad vehicular o la presencia de humo del tabaco, al estar ubicado cerca de un restaurante con área de fumado.

El comportamiento observado de los datos recibidos del nodo (celeste) es esperado; según un estudio realizado en Squinzano, en la provincia de Lecce, Italia [19], donde se midieron los niveles de TVOC de interior y exterior, los niveles usuales se encuentran entre 0.1 ppm y 0.2 ppm en áreas de alta actividad, observable en la Figura 30. Comparando con los momentos de alta actividad registrados en la gráfica de la Figura 29 se puede notar dicho comportamiento.

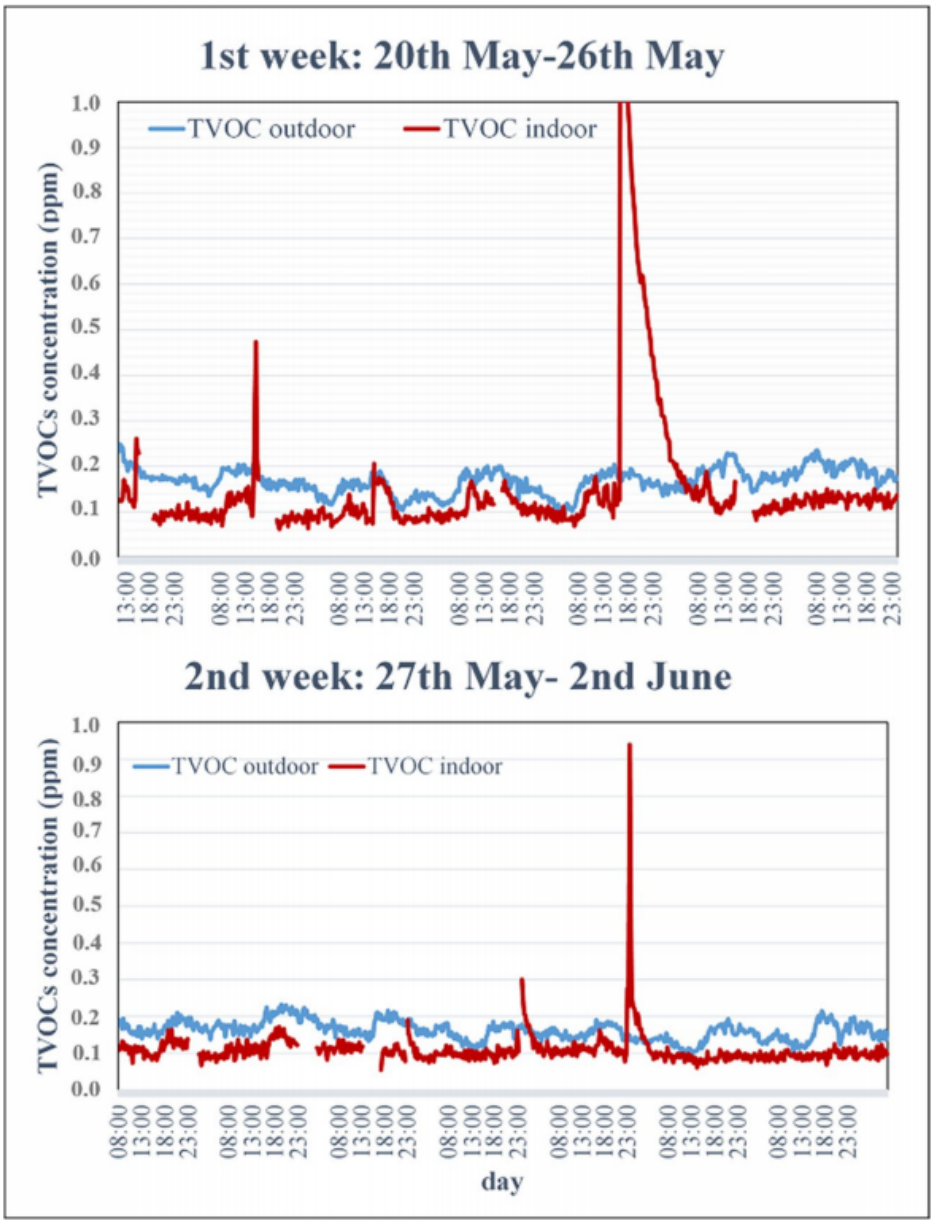

Figura 30: Gráfica de datos de TVOC medidos en Squinzano, Italia. [19]

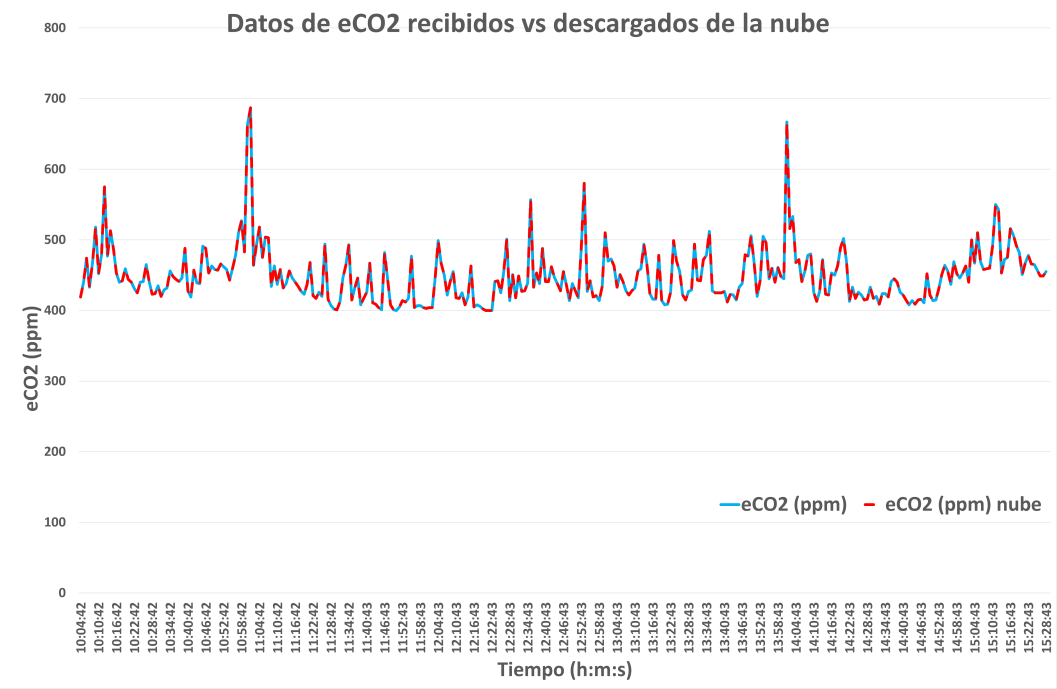

Figura 31: Gráfica de datos de eCO_2 recibidos vs descargados de la nube.

De similar naturaleza se puede observar la gráfica de la Figura 31, donde se miden los niveles de eCO_2. Estos datos poseen un comportamiento estable entre 400 ppm y 500 ppm, esperado en ambientes urbanos promedio según la "Tabla de Niveles de Dióxido de Carbono"de CO2METER [30], con picos de actividad que coinciden con los mismos picos en la gráfica de datos de TVOC, reforzando la fiabilidad del sistema para detectar aumentos en la concentración de gases contaminantes.

Para el material particulado se puede observar su comportamiento en las Figuras 32 para partículas $PM_{2,5}$ y 33 para partículas PM_{10}.

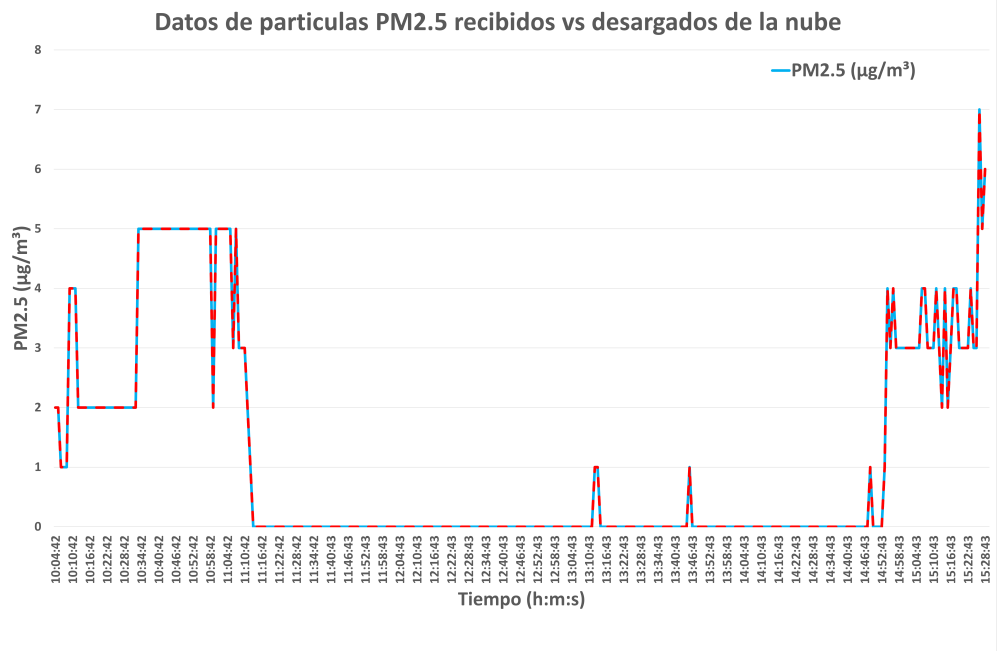

Figura 32: Gráfica de datos de partículas $PM_{2,5}$ recibidos vs descargados de la nube.

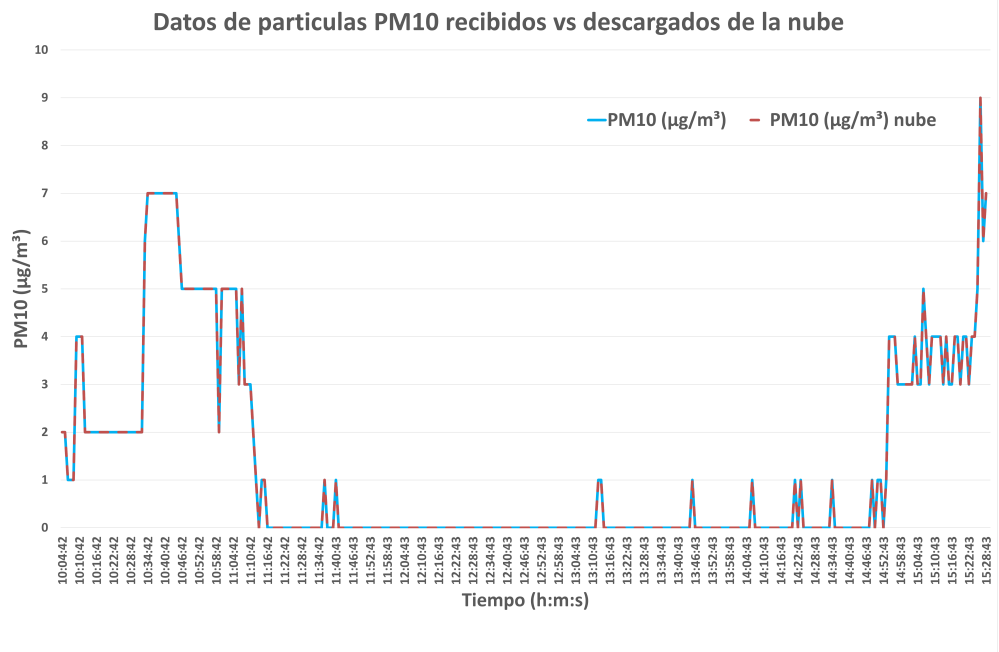

Figura 33: Gráfica de datos de partículas PM_{10} recibidos vs descargados de la nube.

Asimismo, el comportamiento observado por los datos de TVOC y eCO2 se puede notar en las gráficas de materia particulada PM2,5 de la Figura 32 y PM_{10} de la Figura 33, donde se observa un aumento en la concentración de materia particulada detectada en los mismos periodos de actividad elevada.

Finalmente se pueden observar las medidas de AQI según los contaminantes medidos en la Figura 34 y para nivel de ruido en la Figura 35.

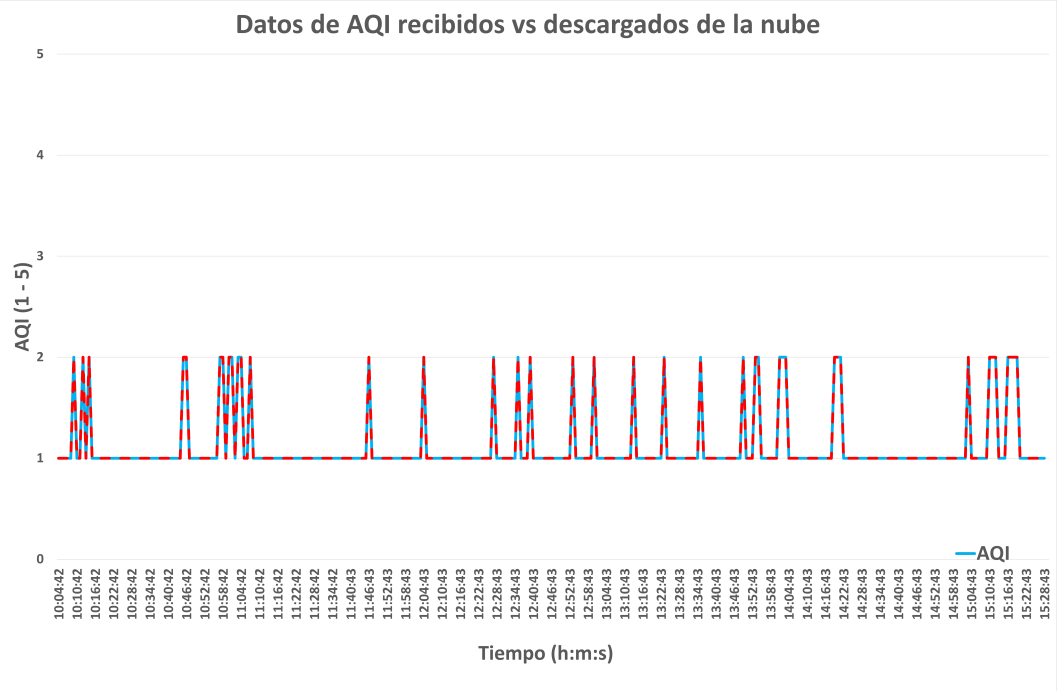

Figura 34: Gráfica de datos de AQI recibidos vs descargados de la nube.

En términos de calidad del ambiente, se puede observar en la gráfica de la Figura 34 un comportamiento de AQI entre nivel 1 (Excelente) y nivel 2 (Bueno) según el estándar del AQI-UBA, con picos concentrados principalmente en los periodos de alta actividad registrados y discutidos.

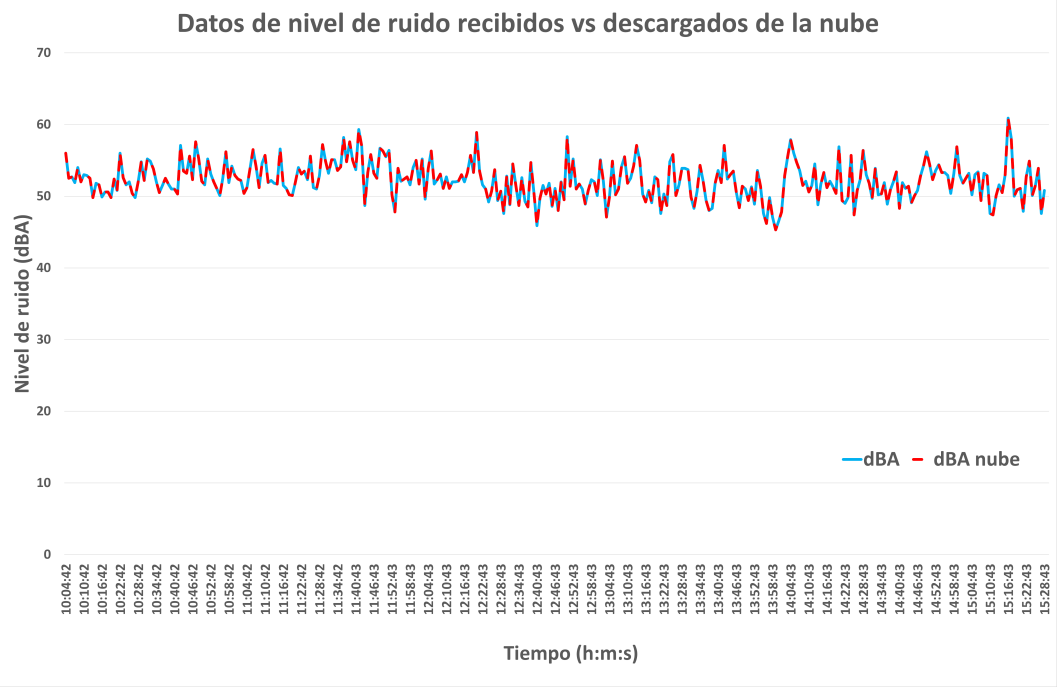

Figura 35: Gráfica de datos de nivel de ruido recibidos vs descargados de la nube.

El nivel de ruido ambiental se encuentra entre 50 dBA y 60 dBA, esperado para un área residencial urbana; según el estudio *Ruido ambiental en áreas verdes urbanas y periurbanas de una microcuenca en Heredia, Costa Rica"* [34], este nivel de ruido es esperado para áreas urbanas de actividad media, aunque se advierte que la presencia prolongada en estos niveles puede provocar irritación y dolores de cabeza.

Según el estándar de calidad de aire alemán AQI-UBA, se tiene un comportamiento consistente entre todas las variables medidas y su relación con el nivel de calidad asignado: cuando se dan picos de TVOC, eCO_2 y material particulado, estos cambios concuerdan con el cambio de estado de AQI de 1 a 2 en la gráfica de la Figura 34, indicando un comportamiento conforme y consistente del sistema.

13.2. Autonomía

Para la realización de esta prueba se inicializaron tanto el nodo de sensado como la central de datos, aunque con esta última solo se pretende verificar la longevidad de vida mediante alimentación por batería del nodo. Se enciende también la central de datos para recibir los datos de los sensores y de integridad de señal, además de verificar una carga de datos adecuada a la nube.

Considerando todos los componentes elegidos para el nodo de sensado, se calculó el consumo de potencia del sistema; para esto se recopiló el consumo de cada componente 19. Con estos valores se realiza el cálculo de consumo basándose en que la batería posee una capacidad de 10,000 mAh y los componentes consumen aproximadamente $\sim$ 393 mA, y se utiliza la ecuación 1 para realizar el cálculo, con un factor de eficiencia del 95 %, para contemplar la posibilidad de que el banco de poder no posea una capacidad completa o se presenten fugas de corriente en alguna parte del sistema.

Componente	**Consumo de corriente**
Adafruit ENS160	$\sim$ 29 mA
Adafruit SHT40	$\sim$ 0.4 μ A
PMS5003	$\sim$ 100 mA
Gravity SEN0232	$\sim$ 14 mA
Heltec WiFi LoRa 32 v3	$\sim$ 250 mA
Total	$\sim$ 393 mA

Tabla 19: Consumo de corriente de los componentes del nodo de sensado.

$$Autonomia = \frac{Capacidad(mAh)}{Consumo(A)} * 95\,\% \tag{1}$$

$$Autonomia = \frac{10000(mAh)}{393(mA)} * 95\,\% = 24{,}17horas \tag{2}$$

$$Autonomia = 24{,}17horas \tag{3}$$

Con lo que se obtiene un resultado de 24.17 horas teóricas de autonomía. Es importante destacar que este es el "peor de los casos"; al tomar medidas y transmitir datos intermitentemente, el consumo real va a ser menor que el teórico.

Para la prueba real se conecta el banco de poder Inui BI-B41 al nodo de sensado y se deja correr continuamente hasta que la batería se descargue por completo. La central

de datos, al obtener una medida del tiempo activo del nodo, además de la interfaz de Arduino IO que permite visualizar la hora en que se recibió el último paquete de datos, permiten identificar el tiempo que el dispositivo se mantuvo funcionando.

Este se encendió a las 9:40 pm un día viernes y se apagó por sí mismo un día domingo a las 12:10 pm. Al realizar esta prueba se obtiene una medida de autonomía para el nodo de sensado de aproximadamente 38 horas y 30 minutos, se puede observar el ultimo paquete de datos recibido en la Figura 36.

```
[RX] seq=2309 node=1 RSSI=-49 dBm SNR=10.8 dB
   T: 22.88 °C | RH: 83.89 % | TVOC: 0.018 ppm | eCO2: 404 | AQI: 1 | PM2.5: 1 | PM10: 1 | dBA: 48.6 | Uptime: 138600 s
Awaiting data...
```

Figura 36: Ultimo paquete de datos recibido.

Usando esta medida se pueden utilizar las mismas ecuaciones para calcular el consumo real del sistema.

$$Consumo = \frac{10000(mAh)}{38{,}5h} * 95\,\% \tag{4}$$

$$Consumo = 246{,}75mA \tag{5}$$

Dada la prueba de autonomía realizada, el resultado de un periodo de actividad de 38 horas y 30 minutos resulta satisfactorio para la toma de datos por periodos cortos de tiempo. El tiempo activo del nodo de sensado presenta más que la autonomía esperada según la ecuación 2; esto indica un uso de recursos más optimizado que el proyectado de 393 mAh, observado en la ecuación 5 consumo del sistema y una eficiencia energética más alta. Para garantizar funcionamiento continuo sin necesidad de muchos cambios de batería se requiere de mayor capacidad en el banco de poder, alimentación por panel solar o una conexión a la red eléctrica directa del país, aunque esta última opción requeriría de un circuito de acondicionamiento.

14. Conclusiones y recomendaciones

14.1. Conclusiones

- La medición de la calidad de aire no debe necesariamente depender del uso de equipo o sensores de alto precio; al optar por medir contaminantes generales causantes del smog (O_3), además de otros compuestos dañinos e insalubres, se puede reducir el costo de fabricación del sistema y dar un enfoque más preventivo a los datos capturados.
- Se logra realizar la captura de datos ambientales y la detección de niveles de calidad de aire en concordancia con el estándar alemán AQI-UBA, comparando los comportamientos esperados según otros estudios y entre los datos medidos, verificando que los cambios y picos en los datos concuerdan entre sí.
- Las métricas de RSSI y SNR permiten entender la fiabilidad y estabilidad del enlace entre el nodo y la central de datos; al realizar pruebas se puede ver cómo un factor de dispersión SF9 hace que el sistema se mantenga estable por encima del umbral de buena señal de -70 dBm, manteniendo también un nivel de SNR por encima de los 10 dB, lo cual indica un buen enlace y una señal íntegra de la cual se reciben todos los paquetes de datos esperados sin pérdidas.
- Adafruit IO demuestra ser una buena elección como plataforma en la nube, permite la fácil carga de datos y con sus capacidades de interfaz gráfica se logra la visualización y personalización de los datos que se le quieren mostrar al usuario, ya sea un indicador puntual o un historial completo de la información recibida por parte del sistema de sensado.
- Los datos que se envían a la nube concuerdan con los datos recibidos por la central de datos en un 100 %; no se presentan errores entre la información que ve el usuario y la que se recibe del nodo de sensado, verificando la fiabilidad de las métricas que se pueden ver o descargar desde la nube.

14.2. Recomendaciones

Para llegar a un punto donde este sistema pueda llevarse a una etapa de piloto es necesario tomar en cuenta las siguientes recomendaciones.

- Usar un valor de factor de dispersión alto incrementa la confiabilidad y estabilidad de la señal recibida, pero también aumenta el tiempo que el canal permanece ocupado, posiblemente reduciendo la capacidad del canal y el número de nodos que podrían comunicarse eficientemente en una posible red de sensado, por lo que se recomienda realizar un análisis y pruebas para definir cuál valor de SF es el más apropiado.
- Es importante verificar las unidades y cifras significativas provenientes de los datos recolectados, ya que la plataforma en la nube no reconoce valores menores a 0.1 para algunas métricas dependiendo de la visualización seleccionada, por lo que podría existir una discrepancia entre los datos recibidos del nodo de sensado y los descargados de la nube. Se recomienda la estandarización de unidades (como de ppb a ppm), el redondeo al valor mínimo más cercano y el uso de una plataforma en la nube con mejor resolución de datos.
- Para la implementación de este sistema en una aplicación urbana permanente se debe considerar el diseño de un encapsulado más robusto, capaz de soportar el clima del área donde se instale cada nodo y de proteger los electrónicos internos sin comprometer la detección de contaminantes y niveles de ruido en el ambiente.
- se recomienda contactar a grupos como IoTCR o FISCR que poseen experiencia desplegando redes LoRaWAN en aplicaciones de campo, esto debido al estado actual de la infraestructura LoRa en Costa Rica. La mayoría de redes en existencia son privadas o establecidas por comunidades interesadas, en términos de madurez este tipo de infraestructura es viable pero requiere apoyo por parte de los gobiernos locales y las comunidades interesadas en establecer nodos en sus localidades.
- Para una instalación permanente del sistema se recomienda solicitar acceso a la red eléctrica del ICE y agregar al nodo de sensado un circuito acondicionador, se puede considerar también la opción de utilizar un panel solar y un banco de energía de mayor capacidad para alargar el tiempo activo del sistema o de manera mas sencilla se podría optimizar la toma de datos para reducir el consumo energético.

15. Abreviaturas

MINAE - Ministerio Nacional de Ambiente y Energía

IMN - Instituto Meteorológico Nacional

OMS - Organización Mundial de la Salud

EPA - Agencia de Protección Ambiental o Environmental Protection Agency

ICCA - Índice Centroamericano de Calidad de Aire

UBA - Agencia Ambiental Federal de Alemania o Umweltbundesamt

AQG - Guía de Calidad de Aire o Air Quality Guide

AQI - Air Quality Index o Índice de Calidad de Aire

O_3 - Ozono troposférico

NO_2 - Dióxido de nitrógeno

NO_X - Óxidos de nitrógeno

SO_2 - Dióxido de azufre

CO - Monóxido de carbono

TVOC - Compuestos Orgánicos Volátiles Totales o Total Volatile Organic Compounds

eCO_2 - Dióxido de carbono equivalente

$PM_{2,5}$ y PM_{10} - Material particulado fino y grueso, respectivamente

PPM - Partes por millón

PPB - Partes por billón

I2C - Circuito Inter-Integrado o Inter-Integrated Circuit

UART - Transmisor/Receptor Asíncrono Universal o Universal Asynchronous Receiver/Transmitter

ADC - Convertidor Analógico-Digital o Analogue-Digital Converter

SPI - Interfaz Periferal Serial o Serial Peripheral Interface

GPIO - Entrada/Salida de Propósito General o General-Purpose Input/Output

LoRa - Larga Distancia o Long Range

IoT - Internet de las Cosas o Internet of Things

RF - Radiofrecuencia

CSS - Chirp Spread Spectrum

BLE - Bluetooth de baja energía o Bluetooth Low Energy

LiPo - Polímero de Litio o Lithium Polymer

SNR - Relación Señal-Ruido o Signal-Noise Ratio

RSSI - Indicador de Intensidad de Señal Recibida o Received Signal Strength Indicator

SF - Factor de Dispersión o Spreading Factor

MQTT - Message Queuing Telemetry Transport

16. Uso de Recursos

A continuación se presenta una lista de los recursos utilizados:

- HELTEC WiFi LoRa 32(V3), ESP32S3 + SX1262 x2
- Módulo Adafruit ENS160 x1
- Módulo Adafruit SHT40 x1
- Sensor PMS5003
- Módulo Gravity SEN0232 x1
- MakerNova 3.7V LiPo Battery 8000mAh x2
- Multímetro
- Cautín
- Flux
- Encapsulado en corte láser o impresión 3D
- Circuito impreso o PCB
- Cables USB a USB tipo-C
- Cables UART
- Cables I^2C
- Cables tipo jumper
- Laptop o computadora de escritorio
- Arduino IDE
- Adafruit IO
- Acceso a internet
- Servicio eléctrico

17. Costos

El presupuesto consiste principalmente en el costo de los componentes utilizados en el desarrollo de la solución. No es necesaria la obtención de licencias, aunque es recomendado obtener una licencia de radioaficionado de la SUTEL para la implementación del sistema en un entorno de uso real.

Recurso	**Costo**
Módulo Adafruit ENS160	$ 25 USD
HELTEC WiFi LoRa 32(V3), ESP32S3 + SX1262 x2	$ 40 USD
Módulo Adafruit SHT40	$ 25 USD
Sensor PMS5003	$ 25 USD
Módulo Sensor Gravity SEN0232	$ 40 USD
Inui BI-B41	$ 40 USD
Encapsulado	$ 0 USD
PCB	$ 0 USD
Total	$ 195 USD

Tabla 20: Desglose del costo de componentes.

Durante el desarrollo de este proyecto también se utilizaron los siguientes servicios:

Servicio	**Costo**
Electricidad	$ 100 USD
Internet	$ 200 USD
Impresión de PCB	$ 0 USD

Tabla 21: Desglose de servicios utilizados.

Es importante también tener un punto de referencia y comparación con otras alternativas similares disponibles en el mercado. Se tiene primeramente el sistema de sensado Open Air (Model O-1PST); este monitor de calidad de aire presenta sensores de material particulado, TVOC, NO_X, CO_2, temperatura y humedad, y utiliza un microcontrolador ESP32-C3-MINI. Esta opción se encuentra a un precio de $ 225 USD. [46]

Finalmente, también se tiene la opción del sistema GAIA A12; este sistema posee sensores de material particulado, CO_2, temperatura y humedad. Esta opción viene con un panel solar y un banco de poder removible para permitir la actividad continua del sistema de sensado. Se puede adquirir esta opción por un costo mínimo de $ 220 USD, con opciones de hasta $ 380 USD que incluyen capacidades LoRa y GPS. [47]

En general, dados el precio de los componentes y la cantidad de parámetros que se están midiendo, el costo total de $ 195 del proyecto se considera aceptable, ya que el sistema diseñado es capaz de proveer más información y presentar más opciones de comunicación entre nodos y la central de datos que opciones de costo similar.

18. Video de funcionamiento

En el siguiente enlace de YouTube se puede observar un video de demostración del funcionamiento del sistema de sensado.

Enlace: `https://www.youtube.com/watch?v=fCcFa11XrcY`

19. Código

En esta sección se presenta el código implementado para el desarrollo del proyecto.

19.1. Nodo de Sensado

19.1.1. Main

Main.ino

```
void setup() {
  Serial.begin(115200);
  while (!Serial) {}

  // buses I2C: Wire para SHT40, Wire1 para ENS160
  Wire.begin(I2C0_SDA, I2C0_SCL);
  Wire.setClock(400000);
  Wire1.begin(I2C1_SDA, I2C1_SCL);
  Wire1.setClock(400000);

  // escaneo de diagnostico
  i2cScanner(Wire,  "Wire");
  i2cScanner(Wire1, "Wire1");

  // SHT40
  if (!sht40.begin(&Wire)) {
    Serial.println("SHT40 not found @ 0x44 on Wire. Check
       wiring.");
    while (1) delay(10);
  }

  // ENS160 con intento en PREF y ALT
  uint8_t ensAddr = ENS160_ADDR_PREF;
  if (!ens160.begin(&Wire1, ensAddr)) {
    ensAddr = ENS160_ADDR_ALT;
    if (!ens160.begin(&Wire1, ensAddr)) {
      Serial.println("ENS160 not found at 0x52 or 0x53 on
         Wire1.");
      while (1) delay(10);
    }
  }
```

```
  Serial.print("ENS160 ready at 0x"); Serial.println(ensAddr,
    HEX);

  // PMS5003 y SEN0232
  pms5003.begin(PMS_UART, PMS_RX_PIN, PMS_TX_PIN, PMS_SET_PIN,
      PMS_RST_PIN);
  sen0232.begin(SEN0232_PIN);

  // LoRa
  loraTx.begin();

  // inicializacion de timers
  lastPrintMs = millis();
  lastPmsPollMs = millis();
  lastSend = millis();
}

void loop() {
  unsigned long now = millis();

  // sondeo rapido del PMS para no perder frames
  if (now - lastPmsPollMs >= PMS_POLL_MS) {
    lastPmsPollMs = now;
    if (pms5003.read()) {
      c_pm25_cf1 = pms5003.pm2_5_cf1();
      c_pm10_cf1 = pms5003.pm10_cf1();
      c_pm25_atm = pms5003.pm2_5_atm();
      c_pm10_atm = pms5003.pm10_atm();
    }
  }

  // impresion periodica y muestreo coherente
  if (now - lastPrintMs >= PRINT_PERIOD_MS) {
    lastPrintMs += PRINT_PERIOD_MS;

    // 1) leer T/RH
    float tC = NAN, rh = NAN;
    if (!sht40.read(tC, rh)) {
      Serial.println("SHT40 read failed");
      return;
```

```
}

// 2) compensar ENS160 y actualizar
ens160.setCompensation(tC, rh);
ens160.update();
if (ens160.hasNewData()) {
  c_aqi  = ens160.getAQI_UBA();
  c_tvoc = ens160.getTVOC_ppb();
  c_eco2 = ens160.geteCO2_ppm();
}

// 3) leer dBA si disponible
float dba_tmp, mv_tmp;
if (sen0232.read(dba_tmp, mv_tmp))
  c_dba = dba_tmp;

// 4) log una sola linea
Serial.print("T: "); Serial.print(tC, 2); Serial.print("
    C ");
Serial.print(" | RH: "); Serial.print(rh, 2); Serial.print
   (" %");
Serial.print(" | AQI: "); Serial.print(c_aqi); Serial.
   print(" ("); Serial.print(aqiStatus(c_aqi)); Serial.
   print(")");
Serial.print(" | TVOC: "); Serial.print(c_tvoc); Serial.
   print(" ppb");
Serial.print(" | eCO2: "); Serial.print(c_eco2); Serial.
   print(" ppm");
Serial.print(" | PM(ATM) 2.5: "); Serial.print(c_pm25_atm)
   ;
Serial.print("  10: "); Serial.print(c_pm10_atm);
Serial.print(" | dBA: "); Serial.println(c_dba, 1);

// 5) acumular para promedios y envio
nSamples++;
sum_tC   += tC;
sum_rh   += rh;
sum_aqi  += c_aqi;
sum_tvoc += c_tvoc;
sum_eco2 += c_eco2;
```

```
    sum_pm25 += c_pm25_atm;
    sum_pm10 += c_pm10_atm;
    sum_dba10+= (uint32_t)(c_dba * 10.0f + 0.5f);
  }

  // envio LoRa periodico con promedios
  if (now - lastSend >= SEND_PERIOD_MS) {
    lastSend += SEND_PERIOD_MS;

    if (nSamples > 0) {
      PayloadV1 p;
      initPayloadV1(p, NODE_ID, seq++);

      double inv = 1.0 / (double)nSamples;
      double atC = sum_tC * inv;
      double aRH = sum_rh * inv;

      p.tC_x100  = (int16_t)(atC * 100.0 + 0.5);
      p.rh_x100  = (uint16_t)(aRH * 100.0 + 0.5);
      p.tvoc_ppb = (uint16_t)((double)sum_tvoc * inv + 0.5);
      p.eco2_ppm = (uint16_t)((double)sum_eco2 * inv + 0.5);
      p.aqi_uba  = (uint8_t)((double)sum_aqi * inv + 0.5);
      p.pm25_atm = (uint16_t)((double)sum_pm25 * inv + 0.5);
      p.pm10_atm = (uint16_t)((double)sum_pm10 * inv + 0.5);
      p.dba_x10  = (uint16_t)((double)sum_dba10 * inv + 0.5);
      p.uptime_s = millis() / 1000UL;     // uptime del nodo en
          segundos

      loraTx.transmit(p);

      // reset de acumuladores
      nSamples = 0;
      sum_tC = sum_rh = 0.0;
      sum_aqi = sum_tvoc = sum_eco2 = sum_pm25 = sum_pm10 =
         sum_dba10 = 0;
    }
  }
}
```

19.1.2. Sensor Adafruit SHT40

SHT40Sensor.h

```
#pragma once
#include <Adafruit_SHT4x.h>

class SHT40Sensor {
public:
  // Inicializa el sensor en el bus I2C (default Wire).
  bool begin(TwoWire *bus = &Wire);

  // Dispara una medicion; devuelve true si lee con exito.
  bool read(float &tC, float &rh);

  // Se obtiene del ultimo valor leido.
  float temperatureC() const { return _tC; }
  float humidityRH()  const { return _rh; }

private:
  Adafruit_SHT4x _sht4; // Driver de Adafruit para SHT40
  float _tC = NAN, _rh = NAN; // Ultimas lecturas (C, %RH)
};
```

SHT40Sensor.cpp

```
#include "SHT40Sensor.h"

bool SHT40Sensor::begin(TwoWire *bus) {
  // begin() retorna false si el dispositivo no responde en el
      bus I2C
  if (!_sht4.begin(bus)) return false;

  // Alta precision: menor ruido a costa de mayor latencia de
     conversion
  _sht4.setPrecision(SHT4X_HIGH_PRECISION);

  // Heater desactivado para no sesgar la temperatura local
  _sht4.setHeater(SHT4X_NO_HEATER);
  return true;
}
```

```
bool SHT40Sensor::read(float &tC, float &rh) {
  sensors_event_t humidity, temp;

  // Se hace la medicion, si falla return false
  if (!_sht4.getEvent(&humidity, &temp)) return false;

  // Actualiza cache interno
  _tC = temp.temperature;
  _rh = humidity.relative_humidity;

  // Entrega el valor de la medicion por referencia
  tC = _tC;
  rh = _rh;
  return true;
}
```

19.1.3. Sensor Adafruit ENS160

ENS160Sensor.h

```
#pragma once
#include <Wire.h>

// No se incluye ScioSense_ENS16x.h aqui para evitar multiples
    definiciones.
// Se declara la clase y se usa un puntero en la
   implementacion.
class ENS160;

class ENS160Sensor {
public:
  // Inicializa driver y arranca medicion estandar
  bool begin(TwoWire *bus, uint8_t i2c_addr);

  // Modelo de polling: actualiza estado/datos y permite saber
      si hay
  // muestra nueva
  void update();
  bool hasNewData();
```

```
  // Resultados procesados por el algoritmo interno del ENS160
  uint8_t  getAQI_UBA();
  uint16_t getTVOC_ppb();
  uint16_t geteCO2_ppm();

  // Compensacion ambiental: escribe T y RH escaladas a
     registros del
  // ENS160
  void setCompensation(float tC, float rh);

private:
  ENS160*  _ens  = nullptr;  // objeto driver (definido en .
     cpp)
  TwoWire* _bus  = nullptr;  // bus I2C en uso
  uint8_t  _addr = 0x52;     // direccion por defecto;
     verifica 0x52/0x53
};
```

ENS160Sensor.cpp

```
#include "ENS160Sensor.h"
#include "ScioSense_ENS16x.h"  // Driver oficial ScioSense
   ENS16x

// --- Notas de datasheet ENS160 ---
// TEMP_IN (0x13): Temperatura en Kelvin * 64  (LSB primero)
// RH_IN   (0x15): Humedad relativa en % * 512 (LSB primero)
// Escribir compensacion mejora sensiblemente la estimacion de
// TVOC/eCO2/AQI.

// Puntero al driver real (definido en el .h)
static constexpr uint8_t REG_TEMP_IN = 0x13;
static constexpr uint8_t REG_RH_IN   = 0x15;

bool ENS160Sensor::begin(TwoWire *bus, uint8_t i2c_addr) {
  _bus  = bus;
  _addr = i2c_addr;

  // Crear el objeto del driver si aun no existe
```

```
  if (!_ens) _ens = new ENS160();

  _ens->begin(_bus, _addr);
  _ens->init();
  _ens->startStandardMeasure(); // Modo de medicion estandar

  return true;
}

// Llama al update() y lee datos desde los registros del
   ENS160 por I2C.
// Debe llamarse periodicamente
void ENS160Sensor::update() {
  if (_ens) _ens->update();
}

// Indica si el ultimo update() detecto una muestra nueva
   disponible.
bool ENS160Sensor::hasNewData() {
  return _ens ? _ens->hasNewData() : false;
}

// Getters que exponen los valores procesados por el algoritmo
    interno:
uint8_t ENS160Sensor::getAQI_UBA() {
  return _ens ? _ens->getAirQualityIndex_UBA() : 0;
}

uint16_t ENS160Sensor::getTVOC_ppb() {
  return _ens ? _ens->getTvoc() : 0;
}

uint16_t ENS160Sensor::geteCO2_ppm() {
  return _ens ? _ens->getEco2() : 0;
}


// Compensacion ambiental (T en C y RH en %) del ENS160.
// El ENS160 espera TEMP_IN = (C + 273.15) * 64, RH_IN = (%RH)
    * 512,
```

```
// formato LSB a MSB.


void ENS160Sensor::setCompensation(float tC, float rh) {
  if (!_bus) return;

  // Limite de RH a [0, 100] para evitar valores erroneos y
     posibles
  overflows

  if (rh < 0.0f)   rh = 0.0f;
  if (rh > 100.0f) rh = 100.0f;

  // Escalado segun datasheet

  const uint16_t temp_in = (uint16_t)((tC + 273.15f) * 64.0f
  + 0.5f); // Kelvin*64

  const uint16_t rh_in   = (uint16_t)((rh) * 512.0f + 0.5f);
     // % * 512

  // --- Temperatura ---

  _bus->beginTransmission(_addr);
  _bus->write(REG_TEMP_IN);                    // Puntero de
     registro
  _bus->write((uint8_t)(temp_in & 0xFF));      // LSB
  _bus->write((uint8_t)(temp_in >> 8));        // MSB
  uint8_t err = _bus->endTransmission();       // 0 = OK

// Si hubo obtencion de datos fallida se reinicia el proceso

  (void)err;

  // --- Humedad Relativa ---

  _bus->beginTransmission(_addr);
  _bus->write(REG_RH_IN);                      // Puntero de
     registro
  _bus->write((uint8_t)(rh_in & 0xFF));        // LSB
```

```
  _bus->write((uint8_t)(rh_in >> 8));          // MSB
  err = _bus->endTransmission();               // 0 = OK

// Si hubo obtencion de datos fallida se reinicia el proceso

  (void)err;
}
```

19.1.4. Sensor Plantower PMS5003

PMS5003Sensor.h

```
#pragma once
#include <Arduino.h>

class PMS;

class PMS5003Sensor {
public:
  // Se asigna valor de -1 a los pines de SET/RST si estan
     conectados
  bool begin(HardwareSerial &uart, int rxPin, int txPin, int
     setPin = -1, int rstPin = -1);

  // Retorna true cuando se recibe y valida un dato completo
  bool read();

  // Se obtiene el ultimo dato medido
  uint16_t pm1_0_cf1() const { return _pm1_cf1; }
  void sleep();
  void wake();
  void reset();

private:
  HardwareSerial* _uart = nullptr; // Puerto serial
  PMS*            _pms  = nullptr; // Driver que parsea datos

  int _rxPin = -1, _txPin = -1, _setPin = -1, _rstPin = -1;

  // Memoria de ultimas mediciones
  uint16_t _pm1_cf1 = 0, _pm25_cf1 = 0, _pm10_cf1 = 0;
```

```
  uint16_t _pm1_atm = 0, _pm25_atm = 0, _pm10_atm = 0;
};
```

PMS5003Sensor.cpp

```
#include "PMS5003Sensor.h"
#include <PMS.h>

#define PMS_BAUD 9600  // Baudrate

// Mantiene un pin en HIGH (inactivo para SET/RESET)
static void holdHighIfPin(int pin) {
  if (pin < 0) return;
  pinMode(pin, OUTPUT);
  digitalWrite(pin, HIGH);
}

// Genera un pulso y espera un tiempo para estabilizacion
static void pulseActiveLow(int pin, uint16_t lowMs, uint16_t
   settleMs) {
  if (pin < 0) return;
  pinMode(pin, OUTPUT);
  digitalWrite(pin, HIGH);
  delay(5);
  digitalWrite(pin, LOW);
  delay(lowMs);
  digitalWrite(pin, HIGH);
  delay(settleMs);
}

bool PMS5003Sensor::begin(HardwareSerial &uart, int rxPin, int
    txPin,
int setPin, int rstPin) {
  _uart   = &uart;
  _rxPin  = rxPin;
  _txPin  = txPin;
  _setPin = setPin;
  _rstPin = rstPin;

  // Asegura estado inactivo en SET/RESET
  holdHighIfPin(_rstPin); // RESET activo en LOW a HIGH =
```

```
    inactivo
  holdHighIfPin(_setPin); // SET activo en LOW a HIGH = modo
    WORK

  // Inicia el puerto serie
  _uart->begin(PMS_BAUD, SERIAL_8N1, _rxPin, _txPin);

  // Limpia residuales de RX
  while (_uart->available()) (void)_uart->read();

  // Crea el parser sobre este puerto
  _pms = new PMS(*_uart);

  return true;
}

// Lee un dato valido.
// Si llega un dato completo, guarda y retorna true; si no,
   false.
bool PMS5003Sensor::read() {
  if (!_pms) return false;

  PMS::DATA d;
  if (_pms->read(d)) {
    _pm1_cf1  = d.PM_SP_UG_1_0;
    _pm25_cf1 = d.PM_SP_UG_2_5;
    _pm10_cf1 = d.PM_SP_UG_10_0;

    _pm1_atm  = d.PM_AE_UG_1_0;
    _pm25_atm = d.PM_AE_UG_2_5;
    _pm10_atm = d.PM_AE_UG_10_0;
    return true;
  }
  return false;
}

// Control de estado si SET/RESET estan conectados
void PMS5003Sensor::sleep() { if (_setPin >= 0)
digitalWrite(_setPin, LOW);} // SET = LOW sleep
void PMS5003Sensor::wake()  { if (_setPin >= 0)
```

```
digitalWrite(_setPin, HIGH);} // SET = HIGH work
void PMS5003Sensor::reset() { pulseActiveLow(_rstPin, 30, 800)
    ; } // Pulso de reset
```

19.1.5. Sensor Gravity SEN0232

SEN0232Sensor.h

```
    #pragma once
#include <Arduino.h>

/*
 * Wrapper simple para el sensor de ruido SEN0232 (salida
    analogica).
 * - begin(pin): guarda el pin ADC a usar.
 * - read(dBA, mV): lee el ADC, calcula mV y estima dBA (se
    hace en el .cpp).
 * - dBA()/mV(): devuelven el ultimo valor calculado (cache
    interno).
 */
class SEN0232Sensor {
public:
  // Asocia el sensor a un pin de ADC (devuelve false si el
     pin no es valido).
  bool begin(int adcPin);

  // Lee el ADC, convierte a mV y estima dBA. Devuelve true si
      todo ok.
  bool read(float &dBA, float &millivolts);

  // Ultimos valores disponibles sin volver a leer
  float dBA() const { return _dba; }
  float mV()  const { return _mv;  }

private:
  int   _pin = -1;   // Pin de entrada analogica (ADC). -1 =
     no inicializado.
  float _dba = NAN;  // Ultima lectura estimada en dB(A).
  float _mv  = NAN;  // Ultima lectura en milivoltios (util
     para calibrar/depurar).
};
```

SEN0232Sensor.cpp

```
    #include "SEN0232Sensor.h"

// Parametros de calibracion simple mV -> dBA
#define DBA_V_ZERO    0.65f   // Voltaje (V) que equivale ~30
   dBA
#define DBA_SLOPE     51.85f  // dB por volt (ajustar con
   calibracion)
#define DBA_MIN       30.0f   // Limite inferior
#define DBA_MAX       120.0f  // Limite superior

// Lee el ADC (32 muestras) y devuelve mV
static float readMillivoltsInternal(uint8_t pin) {
  analogSetPinAttenuation(pin, ADC_11db); // rango mayor
  analogReadResolution(12);               // 12 bits
  uint32_t acc = 0;
  for (int i = 0; i < 32; ++i) acc += analogRead(pin);
  float avg = acc / 32.0f;
  return (avg * 3300.0f) / 4095.0f;       // cuentas -> mV (
     Vref ~3.3V)
}

// Convierte mV a dBA con modelo lineal y saturacion
static float mvToDbA(float mv) {
  float v = mv / 1000.0f;                 // mV -> V
  float dba = DBA_SLOPE * (v - DBA_V_ZERO) + 30.0f;
  if (dba < DBA_MIN) dba = DBA_MIN;
  if (dba > DBA_MAX) dba = DBA_MAX;
  return dba;
}

// Guarda el pin ADC
bool SEN0232Sensor::begin(int adcPin) {
  _pin = adcPin;
  return true;
}

// Lee ADC -> mV, convierte a dBA y devuelve resultados
bool SEN0232Sensor::read(float &dBA, float &millivolts) {
  if (_pin < 0) return false;
```

```
  _mv  = readMillivoltsInternal(_pin);
  _dba = mvToDbA(_mv);
  dBA = _dba;
  millivolts = _mv;
  return true;
}
```

19.1.6. Transmisión de datos por LoRa

LoRaTransmitter.h

```
#pragma once
#include <Arduino.h>
#include "payload.h"

// Clase simple para transmision LoRa (usa RadioLib
   internamente)
class LoRaTransmitter {
public:
  bool begin();                        // inicia radio, true si
     ok
  int  transmit(const PayloadV1&);     // env a payload,
     devuelve codigo RadioLib

private:
  void* _impl = nullptr;               // puntero a objetos
     internos (SPI/Module/SX1262)
};
```

LoRaTransmitter.cpp

```
#include "LoRaTransmitter.h"
#include <SPI.h>
#include <RadioLib.h>

// Pines del SX1262
static const int PIN_LORA_NSS  = 8;
static const int PIN_LORA_DIO1 = 14;
static const int PIN_LORA_NRST = 12;
static const int PIN_LORA_BUSY = 13;
static const int PIN_LORA_SCK  = 9;
static const int PIN_LORA_MISO = 11;
```

```
static const int PIN_LORA_MOSI = 10;

// Parametros RF:

static const float   FREQ_MHZ = 433.175f; // FREQ
static const float   BW_KHZ   = 125.0f;   // BW
static const uint8_t SF       = 9;        // SF
static const uint8_t CR       = 5;        // CR (4/5)
static const uint16_t PREAM   = 8;        // PREAM
static const uint8_t SYNCWD   = 0x12;     // SYNCWD
static const int8_t  TX_DBM   = 14;       // TX dBm

// Internos RadioLib: SPI + Module + SX1262
struct Impl {
  SPIClass spi{FSPI};
  Module   mod{PIN_LORA_NSS, PIN_LORA_DIO1, PIN_LORA_NRST,
     PIN_LORA_BUSY, spi};
  SX1262   radio{&mod};
};

bool LoRaTransmitter::begin() {
  auto* impl = new Impl();
  _impl = impl;

  // SPI dedicado al radio
  impl->spi.begin(PIN_LORA_SCK, PIN_LORA_MISO, PIN_LORA_MOSI,
     PIN_LORA_NSS);

  // FREQ (frecuencia portadora)
  int st = impl->radio.begin(FREQ_MHZ);
  if (st != RADIOLIB_ERR_NONE) return false;

  // Heltec V3.x: DIO2 como RF-Switch
  impl->radio.setDio2AsRfSwitch(true);

  // Ajustes RF: BW / SF / CR / PREAM / SYNCWD / CRC / TX dBm
  impl->radio.setBandwidth(BW_KHZ);          // BW: ancho de
     banda
  impl->radio.setSpreadingFactor(SF);        // SF: factor de
     dispersion
```

```
  impl->radio.setCodingRate(CR);                // CR: tasa de
      codificacion
  impl->radio.setPreambleLength(PREAM);         // PREAM:
      preambulo
  impl->radio.setSyncWord(SYNCWD);              // SYNCWD: sync
      word
  impl->radio.setCRC(true);                     // CRC habilitado
  impl->radio.setOutputPower(TX_DBM);           // TX dBm:
      potencia
  return true;
}

int LoRaTransmitter::transmit(const PayloadV1& p) {
  auto* impl = static_cast<Impl*>(_impl);
  if (!impl) return RADIOLIB_ERR_UNKNOWN;

  // Envia el payload binario completo
  int st = impl->radio.transmit((uint8_t*)&p, sizeof(PayloadV1
      ));

  // Log minimo
  Serial.print("TX seq="); Serial.print(p.seq);
  Serial.print(" bytes="); Serial.print(sizeof(PayloadV1));
  if (st == RADIOLIB_ERR_NONE) Serial.println(" ok");
  else { Serial.print(" fail "); Serial.println(st); }

  return st;
}
```

payload.h

```
#pragma once
#include <Arduino.h>

// Payload compacto (23 bytes) sin padding
#pragma pack(push, 1)
struct PayloadV1 {
  uint8_t  version;     // version de payload
  uint8_t  nodeId;      // ID de nodo
  uint16_t seq;         // secuencia (sequence)
  int16_t  tC_x100;     // temperatura x100
```

```
  uint16_t rh_x100;     // humedad relativa x100
  uint16_t tvoc_ppb;    // TVOC en ppb
  uint16_t eco2_ppm;    // eCO2 en ppm
  uint8_t  aqi_uba;     // AQI (Air Quality Index, UBA)
  uint16_t pm25_atm;    // PM2.5 (ATM, g/m3)
  uint16_t pm10_atm;    // PM10 (ATM, g/m3)
  uint16_t dba_x10;     // nivel sonoro dB(A) x10
  uint32_t uptime_s;    // tiempo encendido (s)
};
#pragma pack(pop)

// Verificacion de tama o exacto
static_assert(sizeof(PayloadV1) == 23, "PayloadV1 size
   mismatch!");
```

19.2. Central de Datos

19.2.1. Main

CentralDatos.ino

```
#include <Arduino.h>
#include <Wire.h>
#include "HT_SSD1306Wire.h"
#include "payload.h"
#include "LoRaReceiver.h"
#include "NetWiFi.h"
#include "AIOClient.h"
#include <WiFi.h>  // estado WiFi

// OLED integrado en Heltec WiFi LoRa 32 V3
#define OLED_ADDR_PRIMARY   0x3C
#define OLED_ADDR_FALLBACK  0x3D
#define OLED_SDA            17
#define OLED_SCL            18
#define VEXT_CTRL           36
#define OLED_RST_PIN        21

// objetos principales: receptor LoRa, enlace WiFi y cliente
   AIO
LoRaReceiver receiver;
```

```
WiFiLink      net;
AIOClient     aio;

// controlador de OLED y estado
static uint8_t g_oled_addr = OLED_ADDR_PRIMARY;
SSD1306Wire oled(g_oled_addr, 100000, OLED_SDA, OLED_SCL,
   GEOMETRY_128_64, OLED_RST_PIN);
static bool oled_ok = false;

// helper I2C para detectar dispositivo en direccion dada
static bool i2cDevicePresent(uint8_t addr) {
  Wire.beginTransmission(addr);
  return (Wire.endTransmission() == 0);
}

// helper simple para saber si hay conexion WiFi
static inline bool wifiConnected() {
  return WiFi.status() == WL_CONNECTED;
}

// pantalla de arranque en el OLED
static void oledShowBoot() {
  if (!oled_ok) return;
  oled.clear();
  oled.setTextAlignment(TEXT_ALIGN_LEFT);
  oled.setFont(ArialMT_Plain_10);
  oled.drawString(0, 0, "Receiver Boot");
  oled.drawString(0, 12, "@433.175 MHz");
  oled.drawString(0, 24, "Awaiting data...");
  // linea de estado WiFi
  oled.drawString(0, 36, wifiConnected() ? "WiFi: OK" : "WiFi:
      --");
  oled.display();
}

// pantalla de espera sin datos
static void oledShowAwaiting() {
  if (!oled_ok) return;
  oled.clear();
  oled.setTextAlignment(TEXT_ALIGN_LEFT);
```

```
  oled.setFont(ArialMT_Plain_10);
  oled.drawString(0, 0, "Awaiting data...");
  // linea de estado WiFi
  oled.drawString(0, 12, wifiConnected() ? "WiFi: OK" : "WiFi:
      --");
  oled.display();
}

// pantalla con info de recepcion correcta
static void oledShowRxOK(uint16_t seq, int16_t rssi, float snr
  ) {
  if (!oled_ok) return;
  oled.clear();
  oled.setTextAlignment(TEXT_ALIGN_LEFT);
  oled.setFont(ArialMT_Plain_10);
  oled.drawString(0, 0,  "RX OK seq=" + String(seq));
  oled.drawString(0, 12, "RSSI: " + String(rssi) + " dBm");
  oled.drawString(0, 24, "SNR:  " + String(snr, 1) + " dB");
  oled.display();
}

void setup() {
  Serial.begin(115200);
  while (!Serial) {}

  // habilita alimentacion Vext para OLED
  pinMode(VEXT_CTRL, OUTPUT);
  digitalWrite(VEXT_CTRL, LOW);
  delay(50);

  // inicia bus I2C del OLED
  Wire.begin(OLED_SDA, OLED_SCL);
  delay(10);

  // reset de OLED por pin dedicado
  pinMode(OLED_RST_PIN, OUTPUT);
  digitalWrite(OLED_RST_PIN, LOW);
  delay(10);
  digitalWrite(OLED_RST_PIN, HIGH);
  delay(10);
```

```
// detecta OLED en 0x3C o 0x3D y configura direccion
   efectiva
bool has3C = i2cDevicePresent(OLED_ADDR_PRIMARY);
bool has3D = (!has3C) && i2cDevicePresent(OLED_ADDR_FALLBACK
   );

if (has3C || has3D) {
  if (has3D) {
    g_oled_addr = OLED_ADDR_FALLBACK;
    // reconstruccion en sitio con direccion alternativa
    new (&oled) SSD1306Wire(g_oled_addr, 100000, OLED_SDA,
       OLED_SCL, GEOMETRY_128_64, OLED_RST_PIN);
  }
  oled.init();
  oled.clear();
  oled.setContrast(255);
  oled_ok = true;
} else {
  oled_ok = false;
  Serial.println("[OLED] Not detected at 0x3C/0x3D (Vext ON,
      RST pulsed).");
}

// muestra pantalla de arranque
oledShowBoot();

// inicializa receptor LoRa; si falla queda detenido
if (!receiver.begin()) {
  Serial.println("LoRa init failed");
  if (oled_ok) {
    oled.clear();
    oled.setTextAlignment(TEXT_ALIGN_LEFT);
    oled.setFont(ArialMT_Plain_10);
    oled.drawString(0, 0, "LoRa init failed");
    oled.display();
  }
  while (1) delay(1000);
}
```

```
  // inicia red WiFi y cliente de nube
  net.begin();
  aio.begin();

  Serial.println("Receiver boot complete");
  Serial.println("Awaiting data...");
}

void loop() {
  static unsigned long lastRxTime = 0; // marca de tiempo del
     ultimo paquete
  static bool showingData = false;     // estado de pantalla
     mostrando datos

  // servicio del cliente de nube (publicaciones, keepalive,
     etc.)
  aio.loop();

  // intento de recepcion de paquete LoRa
  PayloadV1 p;
  if (receiver.receive(p)) {
    // log basico de RF y metadatos del paquete
    Serial.print("[RX] seq="); Serial.print(p.seq);
    Serial.print(" node=");     Serial.print(p.nodeId);
    Serial.print(" RSSI=");     Serial.print(receiver.lastRSSI
       ());
    Serial.print(" dBm SNR="); Serial.print(receiver.lastSNR
       (), 1); Serial.println(" dB");

    // escalado de campos del payload a unidades fisicas
    const float tC      = p.tC_x100 / 100.0f;
    const float rh      = p.rh_x100 / 100.0f;
    const float noiseDb = p.dba_x10 / 10.0f;

    // salida por consola con todos los campos
    Serial.print("   T: ");     Serial.print(tC, 2); Serial.
       print("  C ");
    Serial.print(" | RH: ");    Serial.print(rh, 2); Serial.
       print(" %");
```

```
    Serial.print(" | TVOC: "); Serial.print(p.tvoc_ppb /
       1000.0f, 3); Serial.print(" ppm");
    Serial.print(" | eCO2: "); Serial.print(p.eco2_ppm);
    Serial.print(" | AQI: "); Serial.print((int)p.aqi_uba);
    Serial.print(" | PM2.5: "); Serial.print(p.pm25_atm);
    Serial.print(" | PM10: "); Serial.print(p.pm10_atm);
    Serial.print(" | dBA: "); Serial.print(noiseDb, 1);
    Serial.print(" | Uptime: ");Serial.print(p.uptime_s);
       Serial.println(" s");

    // publica todos los feeds a la nube
    aio.pubT(tC);
    aio.pubRH(rh);
    aio.pubAQI(p.aqi_uba);
    aio.pubNoise(noiseDb);
    float tvoc_ppm = p.tvoc_ppb / 1000.0f;  // ppb a ppm
    aio.pubTVOC(tvoc_ppm);
    aio.pubECO2(p.eco2_ppm);
    aio.pubPM25(p.pm25_atm);
    aio.pubPM10(p.pm10_atm);

    // actualiza OLED con info de recepcion
    oledShowRxOK(p.seq, receiver.lastRSSI(), receiver.lastSNR
       ());
    lastRxTime = millis();
    showingData = true;
  }

  // tras 10 s sin paquetes, volver a pantalla de espera
  if (showingData && millis() - lastRxTime >= 10000UL) {
    showingData = false;
    Serial.println("Awaiting data...");
    oledShowAwaiting();
  }

  // descanso breve del loop
  delay(5);
}
```

19.2.2. Recepción de datos por LoRa

LoRaReceiver.h

```
#pragma once
#include <Arduino.h>
#include "payload.h"

// Receptor LoRa basado en RadioLib (SX1262)
class LoRaReceiver {
public:
  bool begin();                    // inicia radio y modo RX
     continuo
  bool receive(PayloadV1 &p);      // intenta leer un payload;
     true si ok
  unsigned long uptimeSeconds(); // segundos desde boot

  int16_t lastRSSI() const;        // ultimo RSSI dBm
  float   lastSNR()  const;        // ultimo SNR dB

private:
  void* _impl = nullptr;           // puntero vacio a objetos
     internos
  unsigned long _bootMs = 0;       // marca de tiempo de
     arranque
};
```

LoRaReceiver.cpp

```
#include "LoRaReceiver.h"
#include <SPI.h>
#include <RadioLib.h>

// Pines del SX1262 (ajustar a hardware)
static const int PIN_LORA_NSS  = 8;
static const int PIN_LORA_DIO1 = 14;
static const int PIN_LORA_NRST = 12;
static const int PIN_LORA_BUSY = 13;
static const int PIN_LORA_SCK  = 9;
static const int PIN_LORA_MISO = 11;
static const int PIN_LORA_MOSI = 10;
```

```
// Parametros RF

static const float   FREQ_MHZ = 433.175f;
static const float   BW_KHZ   = 125.0f;
static const uint8_t SF       = 9;
static const uint8_t CR       = 5;
static const uint16_t PREAM   = 8;
static const uint8_t SYNCWD   = 0x12;

// Estructura interna: SPI + Module + SX1262 + estado de RX
struct Impl {
  SPIClass spi{FSPI};
  Module   mod{PIN_LORA_NSS, PIN_LORA_DIO1, PIN_LORA_NRST,
     PIN_LORA_BUSY, spi};
  SX1262   radio{&mod};
  volatile bool rxFlag = false; // marcado por ISR cuando
     llega paquete
  int16_t lastRSSI = 0;         // ultimo RSSI
  float   lastSNR  = 0;         // ultimo SNR
};

// Singleton para ISR (RadioLib requiere callback C-like)
static Impl* singletonImplForIsr = nullptr;
static void setRxFlagIsr() {
  if (singletonImplForIsr) singletonImplForIsr->rxFlag = true;
}

bool LoRaReceiver::begin() {
  auto* impl = new Impl();
  _impl = impl;
  _bootMs = millis();

  // Inicia bus SPI para el radio
  impl->spi.begin(PIN_LORA_SCK, PIN_LORA_MISO, PIN_LORA_MOSI,
     PIN_LORA_NSS);

  // Configura frecuencia (FREQ)
  int st = impl->radio.begin(FREQ_MHZ);
  if (st != RADIOLIB_ERR_NONE) return false;
```

```
  // Requerido en Heltec V3.x para manejar RF switch por DIO2
  impl->radio.setDio2AsRfSwitch(true);

  // Ajustes RF: BW, SF, CR, PREAM, SYNCWD y CRC
  impl->radio.setBandwidth(BW_KHZ);
  impl->radio.setSpreadingFactor(SF);
  impl->radio.setCodingRate(CR);
  impl->radio.setPreambleLength(PREAM);
  impl->radio.setSyncWord(SYNCWD);
  impl->radio.setCRC(true);

  // Configura ISR en DIO1 para marcar llegada de paquete
  singletonImplForIsr = impl;
  impl->radio.setDio1Action(setRxFlagIsr);

  // Entra en recepcion continua
  impl->radio.startReceive();
  Serial.println("LoRa receiver ready @433.175 MHz");
  return true;
}

bool LoRaReceiver::receive(PayloadV1 &p) {
  auto* impl = static_cast<Impl*>(_impl);
  if (!impl) return false;

  // Si ISR no marco llegada, no hay datos
  if (!impl->rxFlag) return false;
  impl->rxFlag = false;

  // Buffer de lectura del tamanio exacto del payload
  uint8_t buf[sizeof(PayloadV1)];
  size_t  len = sizeof(buf);
  int state = impl->radio.readData(buf, len);

  // Validar estado y longitud
  if (state == RADIOLIB_ERR_NONE && len == sizeof(PayloadV1))
     {
    // Copiar a estructura de salida
    memcpy(&p, buf, sizeof(PayloadV1));
```

```
    // Capturar metrica de enlace
    impl->lastRSSI = impl->radio.getRSSI();
    impl->lastSNR  = impl->radio.getSNR();

    // Reiniciar recepcion continua
    impl->radio.startReceive();
    return true;
  } else {
    // Si fallo o tamanio inesperado, seguir escuchando
    impl->radio.startReceive();
    return false;
  }
}

// Utilidades de tiempo y ultimas metricas
unsigned long LoRaReceiver::uptimeSeconds() { return (millis()
    - _bootMs) / 1000UL; }
int16_t LoRaReceiver::lastRSSI() const { auto* i = (const Impl
   *)_impl; return i ? i->lastRSSI : 0; }
float   LoRaReceiver::lastSNR()  const { auto* i = (const Impl
   *)_impl; return i ? i->lastSNR  : 0.0f; }
```

19.2.3. Conexión a WiFi

NetWiFi.h

```
#pragma once
#include <Arduino.h>

// Declaracion de la clase WiFiLink que gestiona la conexion
   WiFi
class WiFiLink {
public:
  void begin();              // Metodo para iniciar la
     conexion WiFi
  bool connected() const;    // Metodo para verificar si esta
     conectado
};
```

NetWiFi.cpp

```
#include "NetWiFi.h"
```

```
#include <WiFi.h>
#include "credenciales.h"   // Contiene las credenciales:
    WIFI_SSID y WIFI_PASS

// Inicia la conexion WiFi
void WiFiLink::begin() {
  WiFi.persistent(false);                 // No guarda la red en
     memoria flash
  WiFi.mode(WIFI_STA);                    // Configura el modo
     estacion (STA)
  WiFi.begin(WIFI_SSID, WIFI_PASS);       // Inicia la conexion
     con SSID y contrase a

  unsigned long t0 = millis();            // Marca el tiempo de
     inicio
  while (WiFi.status() != WL_CONNECTED && millis() - t0 < 8000
     UL) {
    delay(200);                           // Espera hasta 8
       segundos para conectarse
  }

  // Verifica si se logro la conexion
  if (WiFi.status() == WL_CONNECTED) {
    Serial.print("[WiFi] Connected: ");
    Serial.print(WiFi.localIP());         // Muestra la IP local
    Serial.print(" RSSI=");               // RSSI: intensidad de
       se al recibida
    Serial.println(WiFi.RSSI());
  } else {
    Serial.println("[WiFi] Not connected (timeout).");
  }
}

// Verifica si hay conexion activa
bool WiFiLink::connected() const {
  return WiFi.status() == WL_CONNECTED;
}
```

credenciales.h

```
#pragma once
```

```
// Nombre de la red WiFi (SSID)
#define WIFI_SSID       "____________"

// Contrase a de la red WiFi
#define WIFI_PASS       "____________"

// Nombre de usuario para Adafruit IO (servicio de IoT en la
   nube)
#define AIO_USERNAME    "____________"

// Clave de acceso para Adafruit IO (API Key)
#define AIO_KEY         "________________________________"
```

19.2.4. Carga de datos a Arduino IO

Adafruit _MQTT _Client.h

```
#pragma once
#include <Arduino.h>

// Clase que maneja la conexion con Adafruit IO usando MQTT
class AIOClient {
public:
  void begin();           // Inicia la conexion MQTT
  void loop();            // Mantiene la conexion activa

  // Metodos para publicar datos ambientales
  bool pubT(float v);           // Temperatura
  bool pubRH(float v);          // Humedad relativa
  bool pubAQI(uint8_t v);       // Indice de calidad del aire
  bool pubNoise(float v);       // Nivel de ruido
  bool pubTVOC(uint16_t v);     // Compuestos organicos
     volatiles
  bool pubECO2(uint16_t v);     // CO2 equivalente
  bool pubPM25(uint16_t v);     // Particulas menores a 2.5
     micras
  bool pubPM10(uint16_t v);     // Particulas menores a 10
     micras

private:
```

```
  void ensureConnected();      // Verifica y reconecta si es
     necesario
}
```

Adafruit_MQTT_Client.cpp

```
#include "AIOClient.h"
#include <WiFi.h>
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"
#include "credenciales.h"

// Configuracion del servidor MQTT de Adafruit IO
#define AIO_SERVER      "io.adafruit.com"
#define AIO_SERVERPORT  1883

// Cliente de red WiFi
static WiFiClient netClient;

// Cliente MQTT con credenciales de Adafruit IO
static Adafruit_MQTT_Client mqtt(&netClient, AIO_SERVER,
   AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);

// Feeds para publicar cada tipo de dato ambiental
static Adafruit_MQTT_Publish f_t    = Adafruit_MQTT_Publish(&
   mqtt, AIO_USERNAME "/feeds/t");
static Adafruit_MQTT_Publish f_rh   = Adafruit_MQTT_Publish(&
   mqtt, AIO_USERNAME "/feeds/rh");
static Adafruit_MQTT_Publish f_aqi  = Adafruit_MQTT_Publish(&
   mqtt, AIO_USERNAME "/feeds/aqi");
static Adafruit_MQTT_Publish f_noise = Adafruit_MQTT_Publish(&
   mqtt, AIO_USERNAME "/feeds/noise");
static Adafruit_MQTT_Publish f_tvoc  = Adafruit_MQTT_Publish(&
   mqtt, AIO_USERNAME "/feeds/tvoc");
static Adafruit_MQTT_Publish f_eco2  = Adafruit_MQTT_Publish(&
   mqtt, AIO_USERNAME "/feeds/eco2");
static Adafruit_MQTT_Publish f_pm25  = Adafruit_MQTT_Publish(&
   mqtt, AIO_USERNAME "/feeds/pm25");
static Adafruit_MQTT_Publish f_pm10  = Adafruit_MQTT_Publish(&
   mqtt, AIO_USERNAME "/feeds/pm10");
```

```
// Verifica si el cliente MQTT esta conectado, si no intenta
   reconectar
void AIOClient::ensureConnected() {
  if (mqtt.connected()) return;

  int8_t ret;
  uint8_t retries = 3;
  while ((ret = mqtt.connect()) != 0) {
    mqtt.disconnect();
    delay(2000);
    if (!--retries) return;
  }
}

// Inicia la conexion MQTT
void AIOClient::begin() {
  ensureConnected();
}

// Mantiene la conexion activa, reconectando si se pierde
void AIOClient::loop() {
  if (!mqtt.connected()) ensureConnected();
}

// Publica un valor numerico en el feed especificado
static bool publishNumber(Adafruit_MQTT_Publish& feed, float
   val) {
  if (!mqtt.connected()) return false;
  return feed.publish(val);
}

// Publica cada tipo de dato ambiental en su feed
   correspondiente
bool AIOClient::pubT(float v)        { ensureConnected();
   return publishNumber(f_t, v); }
bool AIOClient::pubRH(float v)       { ensureConnected();
   return publishNumber(f_rh, v); }
bool AIOClient::pubAQI(uint8_t v)    { ensureConnected();
   return publishNumber(f_aqi, (float)v); }
bool AIOClient::pubNoise(float v)    { ensureConnected();
```

```
    return publishNumber(f_noise, v); }
bool AIOClient::pubTVOC(uint16_t v) { ensureConnected();
    return publishNumber(f_tvoc, (float)v); }
bool AIOClient::pubECO2(uint16_t v) { ensureConnected();
    return publishNumber(f_eco2, (float)v); }
bool AIOClient::pubPM25(uint16_t v) { ensureConnected();
    return publishNumber(f_pm25, (float)v); }
bool AIOClient::pubPM10(uint16_t v) { ensureConnected();
    return publishNumber(f_pm10, (float)v); }
```

20. Referencias

[1] Dr.-Ing. Vega Castillo, P. et al. (2023). *Propuesta de Proyecto "Destinos Turísticos Inteligentes: Tecnologías de Información y Comunicación Aplicadas al Turismo"*.

[2] Dr.-Ing. Vega Castillo, P. et al. (2024). *Propuesta de Proyecto "Destinos Turísticos Inteligentes: Diagnóstico y Recomendaciones para un Plan Estratégico de Turismo Inteligente en Costa Rica"*.

[3] Dr.-Ing. Vega Castillo, P. et al. (2024). *Propuesta de Proyecto "Destinos Turísticos Inteligentes: Cómo Implementar un Destino Turístico Inteligente, Hoja de Ruta y Portafolio de Proyectos"*.

[4] Procuraduría General de la República de Costa Rica. (2025). "*Reglamento sobre Inmisión de Contaminantes Atmosféricos*". [Online]. Disponible en: `http://www.pgrweb.go.cr/scij/Busqueda/Normativa/Normas/nrm_texto_completo.aspx?nValor1=1&nValor2=48141`

[5] Procuraduría General de la República de Costa Rica. (2025). "*Reglamento para el Control de la Contaminación por Ruido*". [Online]. Disponible en: `https://pgrweb.go.cr/scij/Busqueda/Normativa/Normas/nrm_texto_completo.aspx?param1=NRTC&nValor1=1&nValor2=81011&nValor3=106087&strTipM=TC`

[6] Procuraduría General de la República de Costa Rica. (2025). "*Reglamento Sobre Emisión de Contaminantes Atmosféricos Provenientes de Calderas y Hornos de Tipo Directo e Indirecto*". [Online]. Disponible en: `http://www.pgrweb.go.cr/scij/Busqueda/Normativa/Normas/nrm_texto_completo.aspx?param1=NRTC&nValor1=1&nValor2=96890&nValor3=130030&strTipM=TC`

[7] Peña, E. y Legaspi, M. G., Analog Devices (2020). ÜART: A Hardware Communication Protocol."[Online]. Disponible: `https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html`

[8] Google Cloud (2025). "¿Qué es una base de datos en la nube?"[Online]. Disponible: `https://cloud.google.com/learn/what-is-a-cloud-database`

[9] World Health Organization (2021). "*WHO Global Air Quality Guidelines: Particulate Matter (PM2.5 and PM10), Ozone, Nitrogen Dioxide, Sulfur Dioxide and Carbon Monoxide*". [Online]. Disponible: `https://www.who.int/publications/i/item/9789240034228`.

[10] U.S. Environmental Protection Agency (2025). "*Monóxido de carbono en exteriores*". [Online]. Disponible: https://espanol.epa.gov/espanol/monoxido-de-carbono-en-exteriores.

[11] U.S. Environmental Protection Agency, "Ground-Level Ozone Basics". [Online]. Disponible: https://www.epa.gov/ground-level-ozone-pollution/ground-level-ozone-basics

[12] U.S. Environmental Protection Agency, "*El impacto del dióxido de nitrógeno en la calidad del aire interior*". [Online]. Disponible: https://espanol.epa.gov/cai/el-impacto-del-dioxido-de-nitrogeno-en-la-calidad-del-aire-interior

[13] HoloBattery (2025). "Lithium-ion vs. Lithium Polymer Batteries: Which is Better?"[Online]. Disponible: https://holobattery.com/lithium-ion-vs-lithium-polymer-batteries/

[14] U.S. Environmental Protection Agency (2025). "*Sulfur Dioxide Basics*". [Online]. Disponible: https://www.epa.gov/so2-pollution/sulfur-dioxide-basics.

[15] Baral, T. (n.d.). Ünderstanding of LoRa."Medium. [Online]. Disponible: https://medium.com/@arghyabaral/understanding-of-lora-066d89ed7bf4

[16] I. Craggs (2024). "MQTT vs. HTTP: Protocols en IoT e IIoT."[Online]. Disponible: https://www.hivemq.com/blog/mqtt-vs-http-protocols-in-iot-iiot/

[17] U.S. Environmental Protection Agency (2025). "*El impacto de los compuestos orgánicos volátiles en la calidad del aire interior*". [Online]. Disponible: https://espanol.epa.gov/cai/el-impacto-de-los-compuestos-organicos-volatiles-en-la-calidad-del-aire-interior.

[18] Manglai (s.f.). "CO_2 *equivalente: definición y explicación sencilla*". [Online]. Disponible: https://www.manglai.io/glossary/co2-equivalente

[19] Bergomi, A., Mangia, C., Fermo, P., Genga, A., Comite, V., Guadagnini, S., & Ielpo, P. (2024). .ºutdoor trends and indoor investigations of volatile organic compounds in two high schools of southern Italy."[Online]. Disponible: https://link.springer.com/article/10.1007/s11869-024-01509-2

[20] U.S. Environmental Protection Agency (2025). Çlean Air Act Title IV – Noise Pollution". [Online]. Disponible: https://www.epa.gov/clean-air-act-overview/clean-air-act-title-iv-noise-pollution

[21] Wu, J., Texas Instruments (2024). .ª Basic Guide to I2C."[Online]. Disponible: https://www.ti.com/lit/an/sbaa565/sbaa565.pdf

[22] Pico Technology (2025). "Serial Protocol Decoding: I2C."[Online]. Disponible: https://www.picotech.com/library/knowledge-bases/oscilloscopes/serial-protocol-decoding-i2c

[23] ETechnoG (2022). "What is UART Communication? Block Diagram."[Online]. Disponible: https://www.etechnog.com/2022/01/what-is-uart-communication-block-diagram.html

[24] Centro Panamericano de Ingeniería Sanitaria y Ciencias del Ambiente (2025). "*Efectos del cambio climático sobre la calidad del aire urbano y riesgo para la salud pública.*".[Online]. Disponible: https://www.cepis.org.pe/efectos-del-cambio-climatico-sobre-la-calidad-del-aire-urbano-y-riesgo-para-la-salud-publica/

[25] LoRa Alliance (2020). "*LoRaWAN Regional Parameters RP002-1.0.2*". [Online]. Disponible: https://lora-alliance.org/wp-content/uploads/2020/11/RP_2-1.0.2.pdf

[26] The Things Network (n.d.). RSSI and SNR."[Online]. Disponible: https://www.thethingsnetwork.org/docs/lorawan/rssi-and-snr/

[27] Henriksson, R.-M. (2024). Ünderstanding Signal Quality: RSSI, RSRP, and RSRQ..Eseye. [Online]. Disponible: https://www.eseye.com/resources/iot-explained/understanding-signal-quality-rssi-rsrp-and-rsrq/

[28] The Things Network (n.d.). "Spreading Factors."[Online]. Disponible: https://www.thethingsnetwork.org/docs/lorawan/spreading-factors/

[29] Laveyne, J., Van Eetvelde, G., & Vandevelde, L. (2017). .Application of LoRaWAN for smart metering: an experimental verification."[Online]. Disponible: https://www.researchgate.net/publication/319315539_Application_of_LoRaWAN_for_smart_metering_an_experimental_verification/citation/download

[30] CO2Meter (n.d.). Çarbon Dioxide Indoor Levels Chart."[Online]. Disponible: https://www.co2meter.com/blogs/news/carbon-dioxide-indoor-levels-chart

[31] U.S. Environmental Protection Agency (2018). "*Technical Assistance Document for the Reporting of Daily Air Quality – the Air Quality Index (AQI)*", EPA 454/B-18-007. [Online]. Disponible: https://www.airnow.gov/sites/default/files/2020-05/aqi-technical-assistance-document-sept2018.pdf

[32] World Health Organization (2021). .Air quality guidelines: global update 2021: particulate matter, ozone, nitrogen dioxide and sulfur dioxide", WHO/S-DE/PHE/OEH/06.02. [Online]. Disponible: https://iris.who.int/server/api/core/bitstreams/551b515e-2a32-4e1a-a58c-cdaecd395b19/content

[33] Sistema de la Integración Centroamericana (SICA) (2006). "*Estrategia Regional Ambiental Marco (ERAM)*". [Online]. Available: https://www.sica.int/busqueda/busqueda_archivo.aspx?Archivo=libr_9426_1_19062006.pdf

[34] Alfaro-Rojas, D., Portuguez-Brenes, I., Perdomo-Velázquez, H., & Vargas-Masís, R. (2020). Ruido ambiental en áreas verdes urbanas y periurbanas de una microcuenca en Heredia, Costa Rica."*Cuadernos de Investigación UNED*, 12(2), 419–432. [Online]. Disponible: https://dx.doi.org/10.22458/urj.v12i2.2846

[35] IEEE 802 LAN/MAN Standards Committee (2025). ÏEEE 802.11: The Working Group Setting the Standards for Wireless LANs."[Online]. Disponible: https://www.ieee802.org/11/

[36] Proyectos Electrónicos (2025). "*Fundamentos de ESP32: Guía Completa para Principiantes.*"[Online]. Disponible: https://proyectoselectronicos.org/fundamentos-de-esp32-guia-completa-para-principiantes/.

[37] Heltec Automation (2023). "WiFi LoRa 32 V3 Datasheet (HTIT-WB32LA V3.2)."[Online]. Disponible: https://resource.heltec.cn/download/WiFi_LoRa_32_V3/HTIT-WB32LA_V3.2.pdf

[38] Espressif Systems (2025). "*ESP32 Series Datasheet.*"[Online]. Disponible: https://www.espressif.com/en/products/socs/esp32.

[39] ScioSense, "*ENS160 Digital Metal-Oxide Multi-Gas Sensor Datasheet*", Version 1.3, Mar. 2023. [Online]. Available: https://www.sciosense.com/wp-content/uploads/2023/12/ENS160-Datasheet.pdf

[40] Adafruit Industries. *Adafruit ENS160 MOX Gas Sensor - Sciosense CCS811 Upgrade - STEMMA QT / Qwiic*. Disponible en: https://www.adafruit.com/product/5606?srsltid=AfmBOooFdd68sIHfcOgVxe5nSDLlD9RkKAx3qOCJtg4MvAQ6xl1R4kEY.

[41] Adafruit Industries. *Adafruit Sensirion SHT40 Temperature & Humidity Sensor - STEMMA QT / Qwiic*. Disponible en: https://www.adafruit.com/product/4885?srsltid=AfmBOop-jM59BjJheoFCrYccjj-IPTM5Ux1wTBA-D4jHPaH6craB2ail.

[42] ScioSense (2023). .ENS160 Gas Sensor Datasheet."[Online]. Disponible: https://www.sciosense.com/wp-content/uploads/2023/12/ENS160-Datasheet.pdf

[43] DFRobot. *Gravity: Analog Sound Level Meter SKU SEN0232*. Disponible en: https://wiki.dfrobot.com/Gravity__Analog_Sound_Level_Meter_SKU_SEN0232.

[44] Plantower Technology. *PMS5003 — Laser PM2.5 Sensor*. Disponible en: https://www.plantower.com/en/products_33/74.html.

[45] Heltec Automation. *WiFi LoRa 32 (V3), ESP32S3 + SX1262 LoRa Node, Meshtastic and LoRaWAN Compatible*. Disponible en: `https://heltec.org/project/wifi-lora-32-v3/`.

[46] AirGradient (n.d.). .ºutdoor Air Quality Monitors."[Online]. Disponible: `https://www.airgradient.com/outdoor/`

[47] AQICN (n.d.). "GAIA Air Quality Monitoring Stations."[Online]. Disponible: `https://aqicn.org/gaia/`