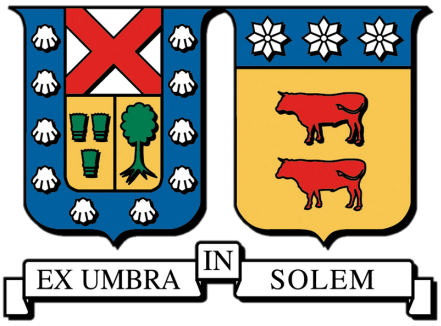
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE



Remote Monitoring of Photovoltaic System Using Drone.

HECTOR ANDRES CEPEDA BLECK

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRONICO

PROFESOR GUÍA: **MOHAMED ABDELHAMID**

PROFESOR CORREFERENTE: **MARCOS ZUÑIGA**

AGOSTO 2025

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): x Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: Remote Monitoring of Photovoltaic System Using Drone

Nombre del candidato(a): Hector Andres Cepeda Bleck

Carrera / Grado: Ingeniería Civil Electrónica

Campus: Casa Central **Departamento:** Electrónica

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo**, Mohamed Abdelhamid** , en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

x El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por **(marcar una opción)**:

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

_____

_____

_____

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

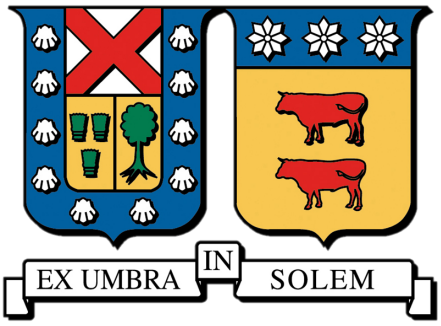
Fecha: 03/11/2025 **Firma:** M. Abdelhamid

Estudiante o Candidato(a):

Fecha: 03/11/2025 **Firma:** _____

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO - CHILE



Remote Monitoring of Photovoltaic System Using Drone.

HECTOR ANDRES CEPEDA BLECK

MEMORIA DE TITULACIÓN PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRONICO

PROFESOR GUÍA: **MOHAMED ABDELHAMID**

PROFESOR CORREFERENTE: **MARCOS ZUÑIGA**

AGOSTO 2025

A mis padres, quienes siempre me apoyaron y confiaron en mi,

A mis amigos, que siempre me motivaron y fueron parte del proceso,

A mis profesores, que me formaron y me han guiado durante este viaje,

A todos los que se han mantenido a mi lado a lo largo de estos años confiando,

Esto es para ustedes.

Agradecimientos

Quiero expresar mi más profundo agradecimiento a quienes hicieron posible la realización de esta memoria.

A mi familia, por su apoyo incondicional, paciencia y confianza durante todo este proceso.

A mi profesor guía, Dr. Mohamed Abdelhamid, por haber sido parte fundamental de este trabajo, entregándome conocimientos, orientación y apoyo constante durante todo el proceso de investigación.

A Vigalab y a mi supervisor de práctica, Julio Contreras , por la oportunidad de aprendizaje y la experiencia profesional que resultó de gran utilidad para la comprensión y desarrollo de esta memoria.

Finalmente, a mis amigos y compañeros de la carrera, por su compañía y motivación constante en este camino universitario.

Resumen

El presente proyecto propone el diseño y validación de un sistema de monitoreo remoto para paneles solares, enfocado en tres escenarios reales: casas, edificios institucionales y granjas solares. El sistema combina sensores ambientales y eléctricos conectados a microcontroladores, que capturan datos críticos del funcionamiento de los paneles y los transmiten hacia la nube utilizando tecnologías como WiFi o LoRa, según el entorno. Una vez en la nube, los datos son visualizados mediante dashboards y pueden activar alertas automáticas si se detectan condiciones anormales (como sobrecalentamiento o pérdida de eficiencia). Ante una alerta, se activa una cámara fija que captura una imagen del panel afectado. Este subsistema de imagen se implementa con una camara usb, con proyección futura de montaje sobre un dron para monitoreo aéreo. La validación del sistema se realiza en dos fases: simulación mediante CupCarbon para evaluar arquitectura y cobertura, y pruebas físicas en laboratorio para validar sensores, comunicación y activación de cámara. El enfoque modular permite adaptar la solución a distintos tipos de instalaciones solares urbanas o rurales, aportando a la optimización y sostenibilidad de la energía solar mediante soluciones IoT escalables en Chile.

Palabras claves: IoT, Paneles fotovoltaicos, Sensores, Cloud, Comunicación LoRa, Captura de imágenes, Dron, alertas

Abstract

This project proposes the design and validation of a remote monitoring system for solar panels, focused on three real-world scenarios: houses, institutional buildings, and solar farms. The system integrates environmental and electrical sensors connected to microcontrollers, which capture critical operational data from the panels and transmit it to the cloud using technologies such as WiFi or LoRa, depending on the environment. Once in the cloud, the data is visualized through dashboards and can trigger automatic alerts when abnormal conditions are detected (such as overheating or efficiency loss). In response to an alert, a fixed camera is activated to capture an image of the affected panel. This imaging subsystem is implemented with a USB camera, with a future projection of mounting it on a drone for aerial monitoring. The system validation is carried out in two phases: simulation with CupCarbon to evaluate architecture and coverage, and laboratory testing to validate sensors, communication, and camera activation. The modular approach allows the solution to be adapted to different types of solar installations, both urban and rural, contributing to the optimization and sustainability of solar energy through scalable IoT solutions in Chile.

Keywords: IoT, Photovoltaic panels, Sensors, Cloud, LoRa communication, Image capture, Drone, Alerts

Glosario

Panel fotovoltaico (PV): Dispositivo que convierte la energía solar en electricidad mediante celdas semiconductoras.

IoT (Internet of Things): Paradigma tecnológico que conecta objetos físicos a internet, permitiendo recopilar, procesar y compartir datos en tiempo real.

Sensor ambiental: Dispositivo que mide variables del entorno como temperatura, humedad relativa e irradiancia solar.

Sensor eléctrico: Dispositivo que mide magnitudes como voltaje, corriente y potencia en un sistema fotovoltaico.

Microcontrolador (MCU): Unidad de cómputo embebida que gestiona sensores y comunicaciones en un nodo IoT.

LoRa: Tecnología de modulación de baja potencia y largo alcance utilizada en comunicaciones inalámbricas de IoT.

LoRaWAN: Protocolo de red basado en LoRa que organiza la comunicación entre nodos IoT y gateways, con gestión de seguridad y escalabilidad.

Gateway (GW): Dispositivo que actúa como intermediario entre los nodos LoRa y la nube, transmitiendo la información recopilada.

Nodo sensor: Dispositivo IoT que integra sensores, MCU y transmisor para capturar y enviar datos del entorno.

ThingSpeak: Plataforma en la nube que permite almacenar, visualizar y analizar datos provenientes de dispositivos IoT.

Dashboard: Interfaz gráfica de usuario para la visualización en tiempo real de datos procesados.

Raspberry Pi: Computador de placa reducida utilizado en este proyecto para procesamiento y disparo de imágenes.

Cámara de monitoreo: Dispositivo de captura de imágenes activado automáticamente ante condiciones anómalas detectadas por el sistema.

Dron: Vehículo aéreo no tripulado que puede portar sensores o cámaras para inspección remota de instalaciones fotovoltaicas.

CupCarbon: Herramienta de simulación de redes IoT que permite evaluar cobertura, consumo y desempeño de tecnologías inalámbricas como LoRa.

Capa de percepción: Nivel del sistema IoT encargado de recopilar datos mediante sensores.

Capa de comunicaciones: Nivel que gestiona la transmisión de datos entre nodos, gateways y servidores.

Capa de aplicación: Nivel que entrega al usuario final visualizaciones, reportes y alertas.

Alerta automática: Notificación generada por el sistema al detectar condiciones críticas como sobrecalentamiento o caída de voltaje.

MQTT: Protocolo ligero de mensajería para IoT que facilita la transmisión de datos entre dispositivos y servidores.

Acrónimos

API: Application Programming Interface

BLE: Bluetooth Low Energy

CEN: Coordinador Eléctrico Nacional

CSV: Comma-Separated Values (formato de archivo)

DNI: Direct Normal Irradiance (Irradiancia Directa Normal)

ERNC: Energías Renovables No Convencionales

GHI: Global Horizontal Irradiance (Irradiancia Global Horizontal)

GW: Gateway

HTTP: Hypertext Transfer Protocol

IoT: Internet of Things

JSON: JavaScript Object Notation (formato de datos)

LoRa: Long Range (modulación de radio de largo alcance)

LoRaWAN: Long Range Wide Area Network

MCU: Microcontroller Unit (Unidad de Microcontrolador)

MQTT: Message Queuing Telemetry Transport

NLOS: Non-Line of Sight (Sin línea de vista)

PV: Photovoltaic (Fotovoltaico)

RF: Radiofrecuencia

RSSI: Received Signal Strength Indicator

SF: Spreading Factor (Factor de dispersión en LoRa)

TS: ThingSpeak

UAV: Unmanned Aerial Vehicle (Vehículo Aéreo No Tripulado)

Wi-Fi: Wireless Fidelity

Índice de figuras

1.1. Topología de capas aplicada al monitoreo fotovoltaico: **percepción**, **adquisición**, **red**, **middleware** y **aplicación**. 7

2.1. Esquema general de un sistema fotovoltaico y sus subsistemas. 12

2.2. Ranking global de electricidad solar en Chile, 2024 [4]. 12

2.3. Arquitectura IoT por capas aplicada al monitoreo energético (esquema conceptual; ver también [11], [12]). 13

2.4. Ejemplo de plataforma IoT multi-planta con capa *edge* y dashboards. *Fuente: adaptado de* [11]. 14

2.5. Topologías LoRa/LoRaWAN (ilustrativo). Para fundamentos de SF/BW/CR y escalabilidad ver [10], [15]. 16

2.6. Comparación cualitativa de tecnologías inalámbricas para telemetría distribuida [12]. 16

2.7. Rango típico de LoRa versus otras tecnologías (ilustrativo); ver consideraciones de sensibilidad/airtime en [10]. 17

2.8. Arquitectura LoRa en planta FV: nodo–gateway–nube [16]. 17

2.9. Esquema de monitoreo FV con LoRaWAN [15]. 18

2.10. LoRaWAN con cifrado y analítica en el borde [18]. 18

2.11. Monitoreo P2P en *junction box* FV [17]. 19

2.12. Arquitectura de monitoreo de salud por panel [19]. 20

2.13. Bloques de un nodo IoT económico para FV [13]. 21

2.14. Ejemplo de *dashboard* multi-sitio con analítica/alertas [11]. 21

2.15. Ejemplo de interfaz móvil para monitoreo [13]. 22
2.16. Inspección autónoma de filas con RGB/térmica [23]. 23

3.1. Topología de capas y tecnologías **adquisición–percepción** (NS), **red** (LoRa), ***middleware*** (gateway/servicios) y **aplicación** (nube). 26
3.2. Conexión de los sensores VEML7700, DHT22 e INA260 a la Raspberry Pi Pico W. 29
3.3. Flujo: Sensores → Pico (JSON) → UART → Heltec TX (LoRa) → Heltec RX (gateway) → nube (ThingSpeak). 34
3.4. Capa de aplicación: ThingSpeak recibe datos del gateway LoRa y entrega tres funciones principales: visualización en paneles, alertas por correo y disparo de captura remota (webcam o dron). 41

4.1. Flujo operativo extremo a extremo 50
4.2. Posiciones angulares del panel en los tres experimentos. 52
4.3. Comparación de potencia P_{pv} por orientación (0°/45°/90°). Se excluye el arranque (20 s). 53
4.4. Posiciones de sombreado parcial aplicadas al panel solar. 54
4.5. Condiciones de limpieza y obstáculos aplicados al panel solar. 55
4.6. **Pérdida de orientación.** Serie temporal de potencia con P_0, t_{on}, $t_{\text{mín}}$ y t_{off} (si aplica). Caída máxima $\sim 42{,}5\,\%$ sin recuperación completa dentro del log. 55
4.7. **Caída a** $0°$ **(1538).** Serie temporal de potencia. Mínimo a ~ 6 s con caída $\sim 27{,}5\,\%$ y recuperación al $95\,\%\,P_0$ en el mismo instante medido. 56
4.8. Comparación normalizada P/P_0 alineada en t_{on} (cuando está disponible). La pérdida de orientación muestra una depresión más profunda y sostenida que el evento de caída a $0°$. 56
4.9. Diagrama de bloques del funcionamiento general del sistema de transmisión y recepción. 57

4.10. Variación real del tamaño de los paquetes transmitidos en 12 secuencias consecutivas: payload MsgPack vs. tamaño total en aire (payload + headers). 58
4.11. Visualización en monitor serial del receptor: reconstrucción, decodificación y estadísticas automáticas del enlace (RSSI, SNR, PRR). . . . 59
4.12. Pantallas OLED de los dispositivos Heltec LoRa TX y RX/Gateway durante el experimento E2. Se observa la transmisión y recepción de paquetes MsgPack fragmentados en dos partes, junto con métricas de desempeño (RSSI, SNR, PRR). 60
4.13. Trayectorias de transmisión en campo real: (a) 25 m, (b) 50 m, (c) 80 m y (d) 150 m, correspondientes a los escenarios experimentales de la etapa E2. 61
4.14. Tasa de recepción de paquetes (PRR) medida en cada escenario experimental. 62
4.15. RSSI promedio observado por escenario experimental. 63
4.16. Flujo de la capa de aplicación: subida de datos, análisis y generación de alertas en `ThingSpeak`. 64
4.17. Dashboards en `ThingSpeak`: gráficos en tiempo real de voltaje, corriente, potencia, temperatura, humedad e irradiancia. 65
4.18. Diagrama de la Capa de Aplicación con integración preliminar (Flask + ngrok + ThingSpeak). 66
4.19. Consola de `ngrok` mostrando las peticiones recibidas desde ThingSpeak hacia el servidor Flask local. 67
4.20. Configuración de la regla *React* en ThingSpeak que dispara el *ThingHTTP* al cumplirse la condición definida. 67
4.21. Configuración del *ThingHTTP* en ThingSpeak apuntando al endpoint expuesto por `ngrok`. 68
4.22. Fotografía capturada automáticamente por la cámara USB al activarse la alerta desde ThingSpeak. 68

4.23. Integración de alertas con dron DJI Tello: flujo nube → servidor Flask → dron, con opciones de disparo manual o automático y transmisión en tiempo real vía WiFi. 70
4.24. Esc. 1 — Topología base (nodo fijo y *gateway* LOS). 72
4.25. Secuencia de recepción (count vs. índice de llegada): incremento monótono sin saltos indica ausencia de pérdidas/duplicados. 72
4.26. PRR por bloque (48 muestras equivalentes): barras en 1.0 muestran entrega uniforme a lo largo de la corrida. 73
4.27. Esc. 2 — *Gateway* móvil (ruta) y ubicación de nodos fijos en el campus. 74
4.28. Paquetes recibidos por *id* en el *gateway* móvil (SINK_82). Se observan cuatro transmisores activos: ID 84, 85, 86 y 95. 75
4.29. Progresión de *seq* recibido vs. orden de llegada, separada por *id*. Las curvas por transmisor muestran tramos de continuidad asociados a ventanas de cobertura a lo largo de la ruta. 75

Índice de cuadros

2.1. Crecimiento de la capacidad solar FV en Chile (2013–2025). 13
2.2. Comparación de sistemas de monitoreo fotovoltaico basados en IoT. . 15
2.3. Sensores clave en monitoreo fotovoltaico (ejemplos reportados en literatura). 20
2.4. Decisiones de diseño de aplicación y respaldo bibliográfico. 22
2.5. Comparación de trabajos relacionados en monitoreo FV con IoT/LoRa y verificación visual (compilación completa). 25

3.1. Conexiones entre la Raspberry Pi Pico W y los sensores 28
3.2. Parámetros de enlace LoRa 34
3.3. Vistas del panel, propósito e indicadores de problema 42
3.4. Patrones multivariantes → diagnóstico probable → acción 44
3.5. Parámetros y recomendaciones 46

4.1. Trazabilidad objetivos ↔ etapas 50
4.2. Símbolos/campos usados en los experimentos. 51
4.3. Potencia por orientación (mediana) y diferencia relativa vs. 90°. . . . 53
4.4. Sombras transitorias: resumen de métricas calculadas sobre los logs del experimento. 54
4.5. Resultados experimentales de entrega LoRa en distintos escenarios. . 62
4.6. Escenarios de simulación en CupCarbon. 71

Índice general

1. Introducción **4**

1.1. Contexto General 4

1.2. Descripción del Problema 5

1.3. Objetivo General 6

1.4. Objetivos Específicos 6

1.5. Análisis del Objetivo 7

1.6. Acercamiento a la Solución 8

1.7. Estructura del Documento 9

2. Marco Teórico **11**

2.1. Sistemas Fotovoltaicos: Principios y Componentes 11

2.2. Arquitectura IoT para monitoreo FV: capas y roles 13

2.3. LPWAN para telemetría FV: LoRa y LoRaWAN 15

2.4. Nodos de borde: microcontroladores y sensores 19

2.5. Middleware y nube: persistencia, visualización y alertas 21

2.6. Verificación visual por evento: cámaras fijas y UAV 23

2.7. Conclusiones del marco integrado 24

2.8. Síntesis comparativa 25

3. Arquitectura del sistema **26**

3.1. Capa de percepción 27

3.2. Capa de comunicaciones 32

3.3. Capa de middleware 36
3.4. Capa de aplicación 40

4. Experimentos, algoritmos y datos **48**
4.1. Objetivos de la validación 48
4.2. Algoritmos implementados (resumen operativo) 49
4.3. Diseño experimental por etapas 50
4.4. E2 — Enlace LoRa y tratamiento del paquete 57
4.5. E3 — Subida de datos a ThingSpeak y visualización en dashboards . . 63
4.6. E4 — Activación de alertas y cámara 66
4.7. Resultados de Simulación CupCarbon 71

5. Conclusiones y trabajo futuro **77**
5.1. Síntesis de la contribución 77
5.2. Discusión de resultados 77
5.3. Limitaciones y amenazas a la validez 78
5.4. Trabajo futuro 79
5.5. Cierre 80

A. Captura de datos a través de los sensores **81**
A.1. A.2 Drivers y gestor de sensores 89
A.1.1. Gestor de sensores (Pico W) 97
A.2. B. Códigos de transmisión y recepción Heltec LoRa 102
A.3. C.1 Códigos de simulación Escenario 2(CupCarbon) 116
A.4. D.1 Alarmas Thingspeak 118

B. Read Channel to Trigger Email 1 **125**

Referencias **128**

Capítulo 1

Introducción

1.1. Contexto General

El crecimiento de la energía solar en Chile ha sido exponencial en la última década. Entre 2013 y 2023, la capacidad instalada fotovoltaica pasó de 8MW a más de 9000MW, impulsada por condiciones únicas de radiación y una caída de costos cercana al 82 % [1], [2]. A fines de 2024, la capacidad total del Sistema Eléctrico Nacional (SEN) alcanzó 34864MW, de los cuales 10507MW correspondieron a fotovoltaica (30,1 %); mientras que las energías renovables no convencionales (ERNC) sin gran hidráulica representaron el 47,5 % del total [3]. Según Ember, Chile se ubica entre los países líderes a nivel mundial en participación solar dentro de su matriz eléctrica, alcanzando cerca de un 20 % de generación a partir de esta fuente en 2023 [4].

Chile cuenta con un gran potencial solar en el desierto de Atacama, donde la irradiación directa normal (DNI) llega a unos 3 500 kWh/m^2 [5], [6]. Esta ventaja ha sido documentada y puesta a disposición del público mediante el *Explorador Solar*, plataforma oficial del Ministerio de Energía [7] con herramientas que permiten calcular y comparar el recurso solar en distintos puntos del territorio, esta herramienta en línea permite estimar el ahorro económico y energético al instalar sistemas fotovoltaicos o térmicos en viviendas, comunidades o industrias en Chile. A nivel de políticas públi-

cas, destacan iniciativas como el programa *Casa Solar*, que entrega cofinanciamiento para la instalación de sistemas residenciales de 1 a 2 kWp, con ahorros estimados entre 150.000 y 300.000 CLP anuales según la capacidad instalada [8]. Asimismo, proyectos comunitarios a lo largo del país como el *Condominio Canquén Norte* en la Región Metropolitana muestran cómo el autoconsumo solar puede reducir entre un 20 % y 30 % el gasto eléctrico familiar [9]. Estos programas reflejan un esfuerzo estatal y privado por masificar la energía solar en sectores residenciales y urbanos, complementando la expansión a gran escala que ocurre en el norte del país.

Sin embargo, el avance también enfrenta importantes desafíos. Uno de los principales es la capacidad de transmisión, dado que gran parte de la generación solar se concentra en el norte, mientras que el mayor consumo ocurre en la zona central. En 2023, en más del 80 % de los días del año la generación renovable superó a la térmica, alcanzando picos de hasta 94,6 % de participación instantánea, lo que genera problemas de vertimiento por falta de flexibilidad en el sistema [2]. A ello se suma la necesidad de incorporar tecnologías de almacenamiento y el problema del ciclo de vida de los paneles. Tanto en instalaciones residenciales como en plantas de gran escala, el monitoreo sigue siendo un reto crítico. Las condiciones geográficas y ambientales dificultan el mantenimiento oportuno, y fallas como acumulación de polvo, sobrecalentamiento local (*hot spots*) o degradación pueden pasar desapercibidas. Por lo tanto, resulta fundamental contar con sistemas de monitoreo remoto que integren sensores, comunicación inalámbrica, análisis en la nube y captura de imágenes (fijas o desde drones), mejorando la operación, la mantenibilidad y la eficiencia de estas instalaciones solares.

1.2. Descripción del Problema

El crecimiento de instalaciones fotovoltaicas en entornos urbanos y rurales ha evidenciado la necesidad de sistemas de monitoreo más eficientes, escalables y automatizados. Actualmente, gran parte de estas instalaciones depende de métodos manuales

o de supervisión básica, lo que dificulta la detección temprana de problemas como suciedad, sombreados, puntos calientes o pérdidas eléctricas. La ausencia de monitoreo confiable y en tiempo real se traduce en pérdidas energéticas, fallas no diagnosticadas y mayores costos de operación y mantenimiento.

Por otro lado, las soluciones disponibles en el mercado suelen ser costosas o de elevada complejidad técnica, lo que limita su aplicación en proyectos de pequeña y mediana escala, como viviendas sociales, edificios institucionales o granjas solares. Esta brecha tecnológica restringe el acceso a herramientas de supervisión inteligente a numerosos usuarios y comunidades que podrían beneficiarse de ellas.

En este escenario, surge la necesidad de un **sistema modular y de bajo costo** que integre IoT, sensado eléctrico/ambiental y análisis en la nube, capaz de emitir **alertas multivariantes** y **verificación visual** (cámara fija o UAV) para acelerar el diagnóstico y reducir pérdidas.

1.3. Objetivo General

Con el fin de aportar en el monitoreo de sistemas fotovoltaicos y ofrecer soluciones modulares, escalables y de bajo costo, **este trabajo propone el diseño y validación de un sistema de monitoreo remoto basado en variables eléctricas y ambientales, tecnologías IoT y LoRa, capaz de visualizar datos críticos en la nube, generar alertas automáticas y activar mecanismos de inspección visual remota** .

1.4. Objetivos Específicos

- Diseñar un prototipo funcional que integre sensores de temperatura, humedad, irradiancia, corriente conectados a un Microcontrolador alimentado por batería externa.

- Implementar comunicación inalámbrica mediante *LoRa* para transmisión de da-

tos hacia un *gateway LoRa* con acceso a Internet.

- Configurar una plataforma para almacenamiento, visualización en tiempo real y generación de alertas automáticas.
- Incorporar un módulo de captura de imágenes como respuesta a condiciones críticas, siguiendo enfoques de monitoreo PV e inspección de imágenes con UAV.
- Validar desempeño (*cobertura/capacidad*) y pruebas de laboratorio/campo (PRR, latencia, consumo).
- Analizar escalabilidad (múltiples nodos/gateways) y pautas de despliegue para edificios o granjas solares.

1.5. Análisis del Objetivo

El sistema propuesto se apoya en una arquitectura IoT de **cinco capas**: **percepción**, **adquisición**, **red**, ***middleware*** y **aplicación**. Los **nodos sensores (NS)** capturan variables de operación y ambiente; la **capa de red** utiliza LoRa/LoRaWAN por su bajo consumo y gran alcance [10]; el ***middleware*** integra y enruta datos hacia la nube; y la **capa de aplicación** ofrece *dashboards* y alertas.

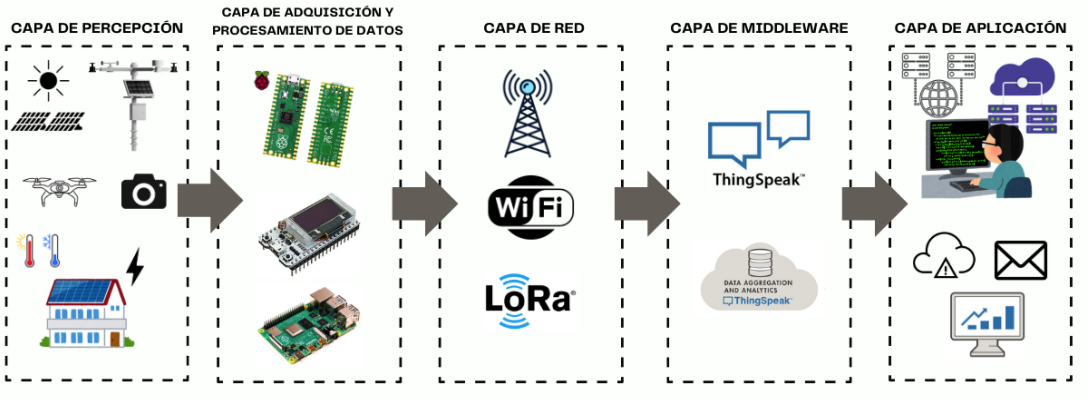


Figura 1.1: Topología de capas aplicada al monitoreo fotovoltaico: **percepción**, **adquisición**, **red**, **middleware** y **aplicación**.

Con base en esta arquitectura, los hitos del desarrollo son:

1. Seleccionar sensores, microcontroladores, módulos de radio y elementos de captura visual.
2. Validar el sistema en laboratorio y en simulaciones de cobertura/capacidad con CupCarbon.
3. Diseñar el firmware para adquisición, empaquetado y transmisión (NS), y recepción/reenrutamiento (gateway).
4. Configurar y probar el enlace LoRa y sus parámetros.
5. Integrar la plataforma en la nube (ThingSpeak) y definir umbrales/alertas.
6. Incorporar el módulo de captura visual por evento cámara RGB (USB/CSI).
7. Definir escenarios de prueba y métricas de evaluación (latencia, confiabilidad, escalabilidad, consumo energético).
8. Realizar la validación integral del sistema, comprobando la interacción de todos los módulos en conjunto (sensores, red, nube y captura visual).
9. Analizar los resultados y proyectar la escalabilidad del sistema hacia instalaciones de mayor tamaño (edificios o granjas solares).

1.6. Acercamiento a la Solución

Como se expuso en el planteamiento del problema, el rápido crecimiento de instalaciones fotovoltaicas en Chile ha puesto en evidencia la necesidad de sistemas de monitoreo más accesibles, escalables y confiables. Actualmente, la supervisión se realiza mayoritariamente con métodos manuales o con soluciones comerciales de alto costo, lo que deja fuera a proyectos de pequeña y mediana escala, tales como viviendas sociales, edificios institucionales o granjas solares de baja potencia. Esto genera

pérdidas energéticas y mayores costos de operación, además de limitar el acceso de comunidades a tecnologías de monitoreo inteligente.

Frente a esta problemática, se propone un sistema modular de monitoreo remoto basado en tecnologías IoT y comunicación LoRa, que ofrezca una alternativa de bajo costo y alto alcance para entornos urbanos y rurales. El acercamiento busca integrar en una sola plataforma la medición de variables eléctricas y ambientales de los módulos fotovoltaicos, junto con capacidades de análisis en la nube y generación de alertas automáticas. De esta forma, los usuarios pueden contar con información en tiempo real para tomar decisiones oportunas y reducir pérdidas por fallas o ineficiencias.

Un aspecto diferenciador de la solución planteada es la incorporación de mecanismos de inspección visual remota, que permiten complementar la información de sensores con evidencia gráfica de los eventos detectados. Esto puede lograrse mediante una cámara fija instalada en el sistema o a través de la coordinación con drones de bajo costo, lo que amplía la capacidad de diagnóstico y reduce la necesidad de inspecciones presenciales frecuentes.

En síntesis, este acercamiento propone una arquitectura IoT de cinco capas (adquisición, percepción, red, middleware y aplicación), donde cada componente cumple un rol clave para garantizar la confiabilidad del sistema. La solución busca ser lo suficientemente flexible para validar su funcionamiento en escenarios de pequeña escala y, al mismo tiempo, proyectarse hacia aplicaciones de mayor envergadura, como edificios o plantas solares.

1.7. Estructura del Documento

El documento se organiza de la siguiente forma:

- **Capítulo 1 — Introducción:** contexto del sector FV en Chile, planteamiento del problema, objetivo general y específicos, análisis del objetivo y acercamiento a la solución.

- **Capítulo 2 — Marco Teórico & Estado del Arte:** fundamentos de sistemas FV y de arquitecturas IoT por capas; tecnologías LPWAN (LoRa/LoRaWAN), nodos y sensores; middleware y nube; verificación visual por evento, y *síntesis comparativa* de trabajos relevantes.

- **Capítulo 3 — Arquitectura del sistema:** diseño de las capas (percepción, comunicaciones, middleware y aplicación), decisiones de HW/SW, flujos E2E y parámetros operativos.

- **Capítulo 4 — Experimentos, algoritmos y datos:** plan de validación E1–E4, métricas, instrumentación, plantillas de resultados y análisis de desempeño extremo a extremo.

- **Capítulo 5 — Resultados:** resultados de *simulación* (CupCarbon) y su comparación entre escenarios; lectura de PRR, RSSI/SNR, consumo y observaciones para escalabilidad.

- **Capítulo 6 — Conclusiones y trabajo futuro:** grado de cumplimiento de objetivos, síntesis de desempeño y proyección de líneas futuras.

Capítulo 2

Marco Teórico

2.1. Sistemas Fotovoltaicos: Principios y Componentes

La energía solar fotovoltaica convierte directamente la radiación solar en electricidad mediante módulos de silicio. Estos sistemas pueden conectarse a la red (*on-grid*) o funcionar de forma autónoma (*off-grid*), siendo aplicables en hogares, instituciones y plantas industriales. Un sistema FV típico se compone de: **paneles solares** que convierten radiación en corriente continua (CC), **inversores** que transforman la CC en corriente alterna (CA), **estructuras de soporte**, **cableado y protecciones**, y **sistemas de monitoreo** que permiten identificar fallas. Entre los principales factores que afectan el rendimiento destacan la acumulación de polvo, el sombreado parcial, la aparición de *hot spots* y fallos eléctricos internos. Estos fenómenos refuerzan la necesidad de contar con sistemas de monitoreo confiables

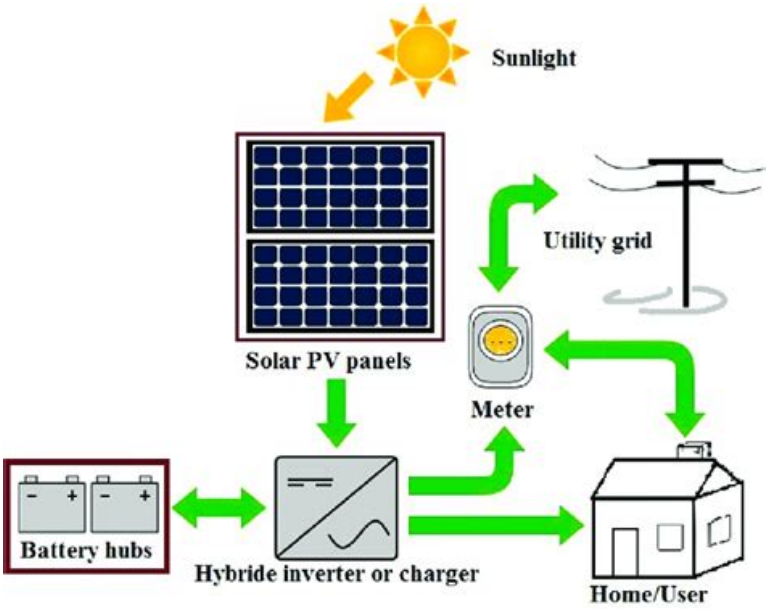


Figura 2.1: Esquema general de un sistema fotovoltaico y sus subsistemas.

En Chile, el recurso solar es excepcional: en el Desierto de Atacama la irradiancia supera los 2.500 kWh/m^2 anuales, ubicando al país entre los líderes mundiales en potencial fotovoltaico [1]. Actualmente más del 90 % de la capacidad solar instalada se concentra en el norte grande, con más de 10 GW operativos hacia 2025 [2], [4].

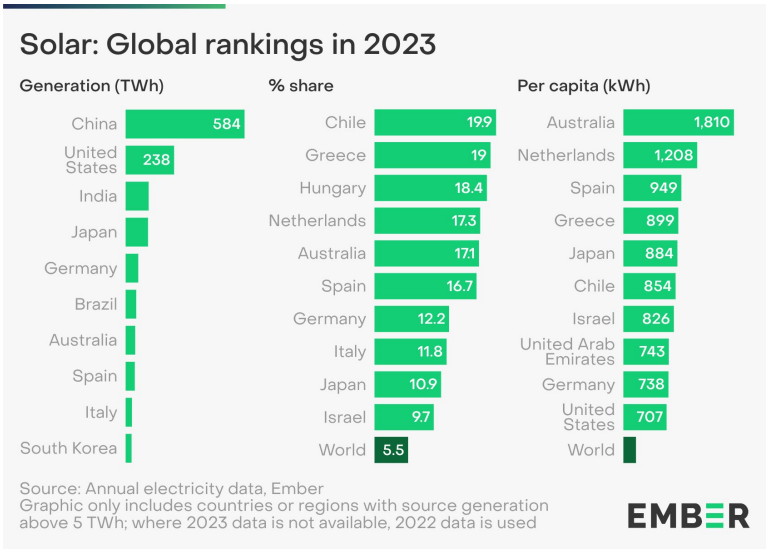


Figura 2.2: Ranking global de electricidad solar en Chile, 2024 [4].

Cuadro 2.1: Crecimiento de la capacidad solar FV en Chile (2013–2025).

Año	Capacidad instalada [MW]
2013	291
2017	1,000 - 900
2020	3,104
2023	10,507
2025	16,804+

2.2. Arquitectura IoT para monitoreo FV: capas y roles

Marco teórico. Una arquitectura IoT típica se organiza en cuatro capas: **adquisición** (sensores y MCU), **comunicaciones** (enlace inalámbrico), **middleware** (gateway/servicios) y **aplicación** (nube: almacenamiento, gráficos y alertas). Una capa de **borde** (*edge*) descarga tareas ligeras (sellado temporal, $P{=}VI$, banderas de calidad) mientras la nube concentra agregación y analítica.

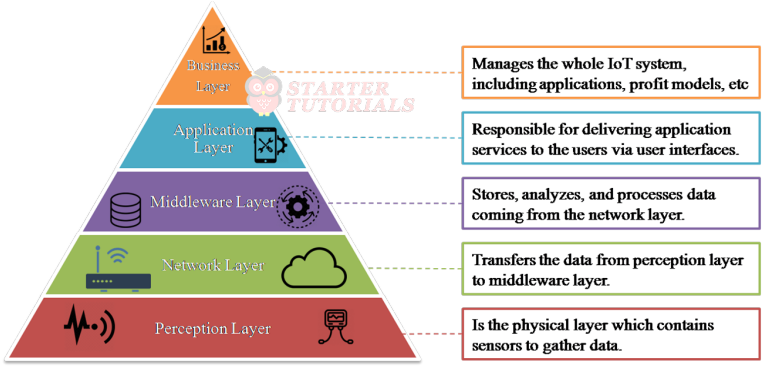


Figura 2.3: Arquitectura IoT por capas aplicada al monitoreo energético (esquema conceptual; ver también [11], [12]).

Evidencia complementaria. Plataformas reales multi-sitio y EMS reportan beneficios consistentes del enfoque por capas y del *edge*: *escalabilidad* y *tolerancia a*

fallos [11]-[13]. En [11] se añade una capa *edge por planta* y se integran analítica de predicción (BiLSTM con R^2=0.9664, MAPE 0.1982) y detección de anomalías (Isolation Forest $95.17\,\%$ de exactitud), con *dashboards* multi-sitio. [12] consolida una pila de cinco capas con *backend híbrido* (*on-premise*/nube) para continuidad operativa y notificaciones centralizadas.

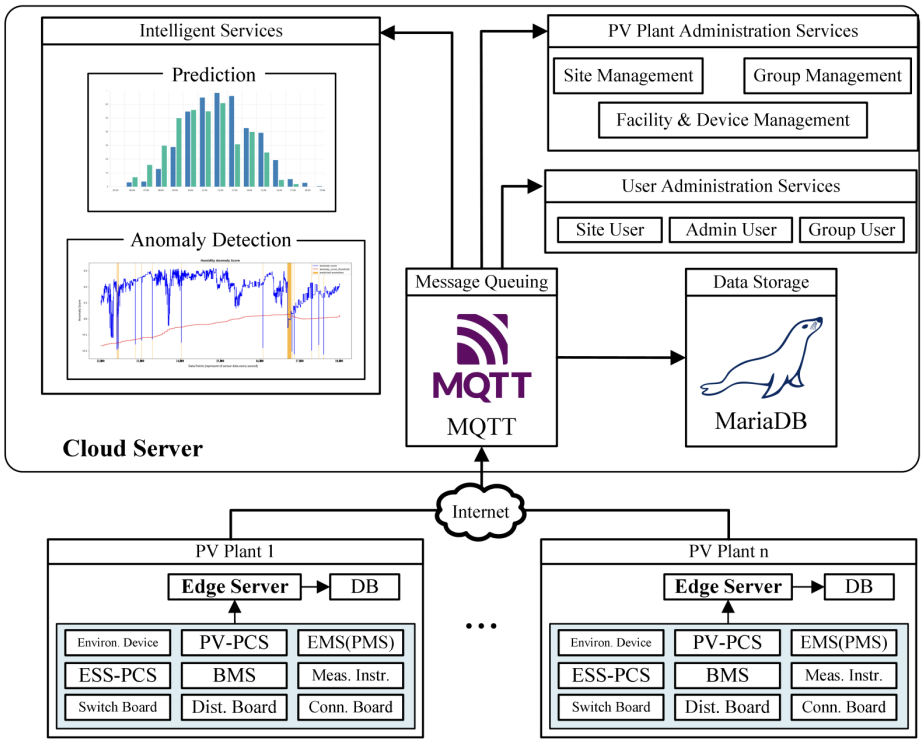


Figura 2.4: Ejemplo de plataforma IoT multi-planta con capa *edge* y dashboards. *Fuente: adaptado de* [11].

Cuadro 2.2: Comparación de sistemas de monitoreo fotovoltaico basados en IoT.

Referencia / Título	**Año**	**Data log system**	**Monitoreo/SCADA**	**Tx datos (LAN)**	**Tx datos (Internet)**
Remote GSM module monitoring and photovoltaic system control **ref16**	2014	PIC18F4550 MCU	LabVIEW	Cableado	GSM
A low-cost solar generation monitoring system suitable for IoT **ref17**	2017	Raspberry Pi	Web APP	ZigBee	N/A
Online monitoring system of PV array based on IoT **ref18**	2017	DSP-TMS320F28335 / Raspberry Pi	Web APP	ZigBee / Wi-Fi	Cableado
An Alternative IoT Solution Based on LoRa for PV Power Plants **ref19**	2019	Arduino / Raspberry Pi	N/A	LoRa	Cableado
IoT Application for Real-Time Monitoring of Solar Home Systems **ref20**	2019	Arduino UNO	ThingSpeak	Cableado	3G
A Real-time Monitoring System Based on ZigBee and 4G **ref21**	2020	Cloud server	Web APP	ZigBee	4G
A Low-Cost IoT System for Real-Time Monitoring (Smart Grid) **ref22**	2021	Heltec Wi-Fi LoRa 32	Web APP	LoRa / Wi-Fi	Cableado
Proposed system	–	Central / Cloud servers	Web APP	Cableado / Wi-Fi / IR / BLE	Cableado / 4G LTE

2.3. LPWAN para telemetría FV: LoRa y LoRaWAN

Marco teórico. LoRa (CSS) y LoRaWAN operan en bandas ISM (915 MHz en Chile [14]) con muy bajo consumo y gran alcance. El desempeño depende de **SF/BW/CR**: mayor SF incrementa sensibilidad (y airtime), BW ajusta robustez/velocidad y CR protege contra errores [10].

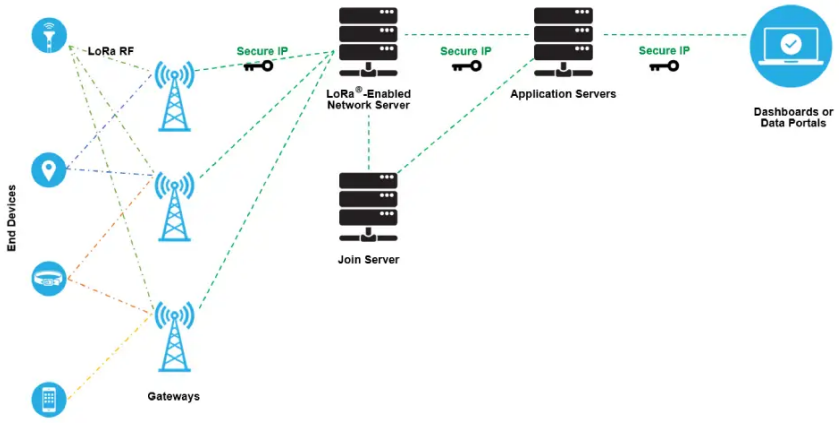


Figura 2.5: Topologías LoRa/LoRaWAN (ilustrativo). Para fundamentos de SF/BW/CR y escalabilidad ver [10], [15].

Evidencia complementaria. En FV, se han reportado coberturas > 1 km con bajo consumo [16] y PRR $> 90\,\%$ en pruebas de campo variando SF/BW [15]. Para pilotos/localizaciones específicas, enlaces LoRa *punto a punto* simplifican despliegue manteniendo confiabilidad (monitoring en *junction box*) [17]. A nivel de ecosistema, EMS comparan tecnologías (ZigBee, WiFi, Celular, NB-IoT, LoRa) según cobertura, tasa y consumo [12].

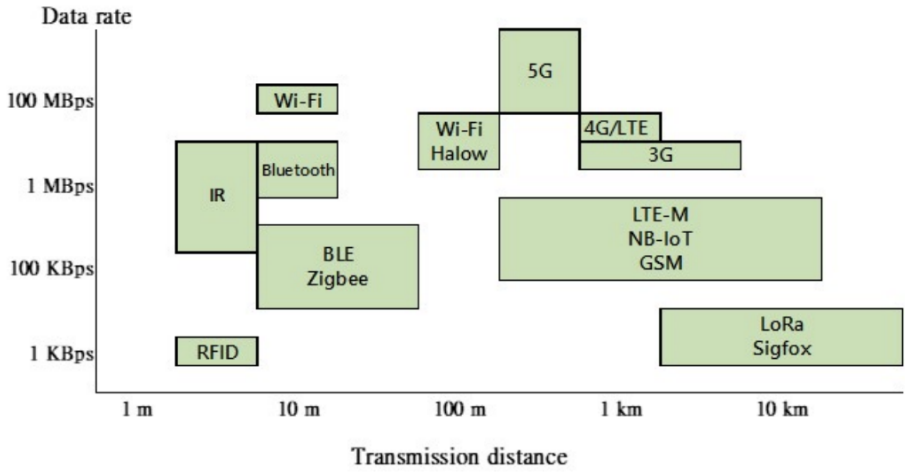


Figura 2.6: Comparación cualitativa de tecnologías inalámbricas para telemetría distribuida [12].

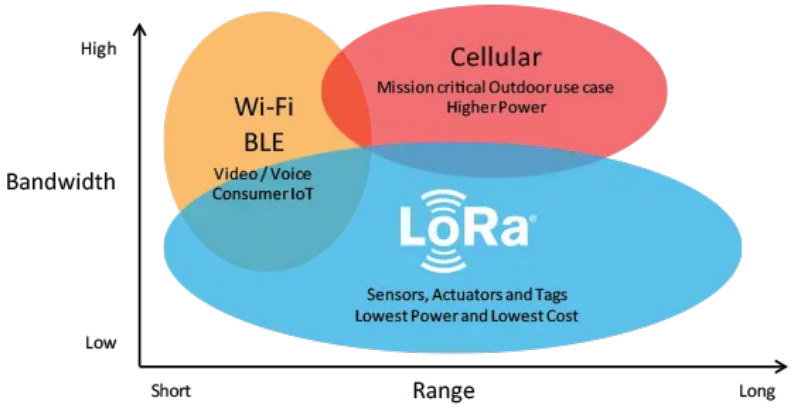


Figura 2.7: Rango típico de LoRa versus otras tecnologías (ilustrativo); ver consideraciones de sensibilidad/airtime en [10].

Arquitectura LoRa nodo–gateway–nube en planta FV. Para plantas conectadas a red, LoRa punto a punto desde *nodes* a un gateway reduce cableado y costo manteniendo cobertura (telemetría periódica) [16].

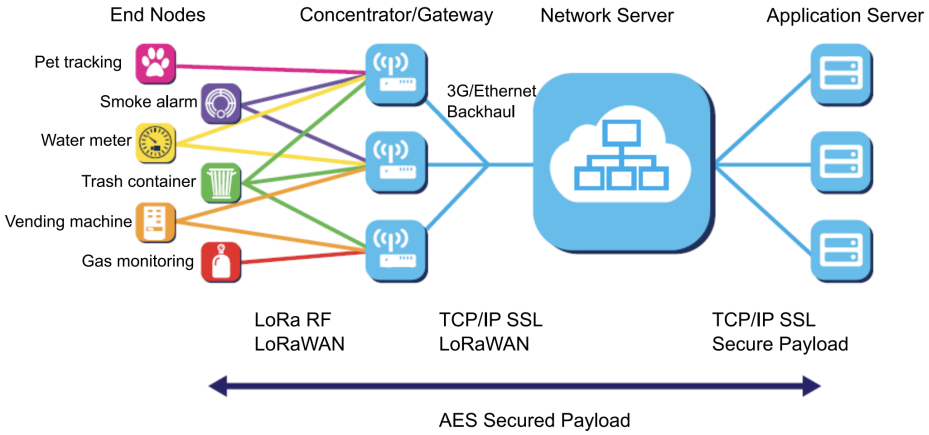


Figura 2.8: Arquitectura LoRa en planta FV: nodo–gateway–nube [16].

Esquema LoRaWAN cuando se requiere escalabilidad. Cuando crece el número de nodos y la cobertura, LoRaWAN (servidor de red + gateways) ofrece gestión y *duty cycle* coordinado, manteniendo PRR $> 90\,\%$ en campo con configuración SF/BW adecuada [15].

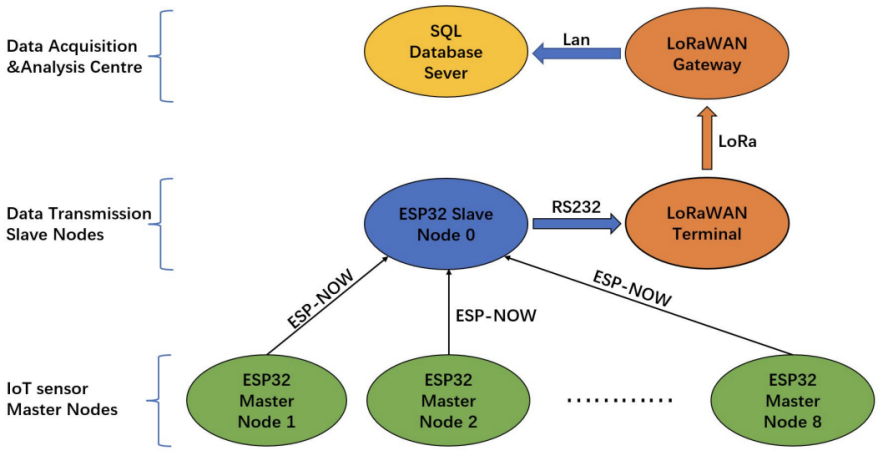


Figura 2.9: Esquema de monitoreo FV con LoRaWAN [15].

Seguridad y analítica en el borde (LoRaWAN).

Para reducir latencia y dependencia de la nube, [18] ejecutan clasificación ligera (SVM) en el *edge* con cifrado extremo a extremo, logrando precisión $> 90\,\%$ en detección temprana y manteniendo bajo consumo.

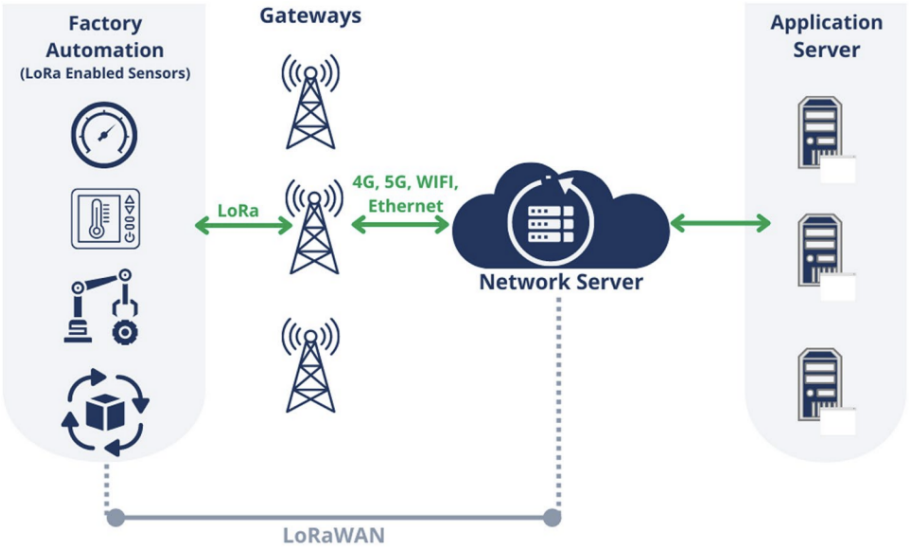


Figura 2.10: LoRaWAN con cifrado y analítica en el borde [18].

Variación P2P en *junction boxes*. Para vigilancia térmica de diodos y conexiones internas, [17] integran LoRa P2P directamente en la *junction box*, útil en pilotos o *strings* extensos donde la simplicidad prima sobre la gestión de red.

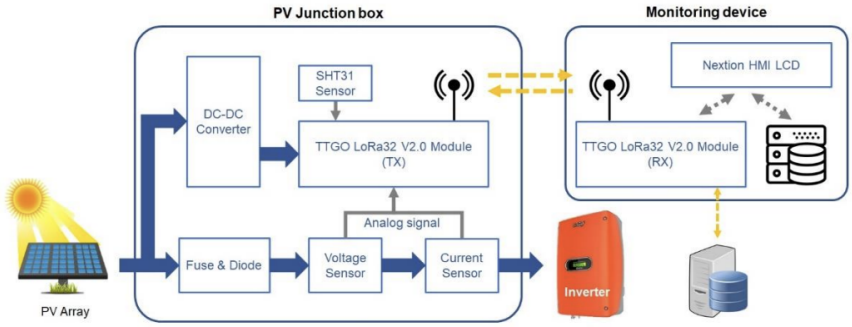


Figura 2.11: Monitoreo P2P en *junction box* FV [17].

2.4. Nodos de borde: microcontroladores y sensores

Marco teórico. Para diagnóstico FV a baja cadencia, el conjunto mínimo relevante es $\{V_{pv}, I_{pv}, P_{pv}, \text{lux}, T, RH\}$, suficiente para explicar variaciones de producción por recurso y condición térmica/ambiental. Buses digitales (I^2C/1-Wire) reducen errores analógicos y simplifican integración.

Evidencia complementaria. Sistemas de **bajo costo** con sensores digitales han demostrado suficiencia para monitoreo operativo y estimación de potencia (p. ej., Random Forest $\sim$87 % de exactitud) [13], [19]. En el extremo avanzado, monitoreo a nivel de panel añade trazado I–V/EL y *reconfiguración activa* del *string* [20].

Cuadro 2.3: Sensores clave en monitoreo fotovoltaico (ejemplos reportados en literatura).

Variable	**Ejemplo de sensor**	**Uso en diagnóstico**
Temperatura/humedad	DHT22, SHT31	Identificación de deriva térmica y condiciones ambientales
Iluminancia	VEML7700	Detección de sombreado o variación del recurso; calibrable a W/m^2
Corriente	INA260	Seguimiento de producción energética y detección de anomalías
Voltaje	INA219, INA260	Identificación de caídas anómalas de tensión en módulos o strings

Monitoreo de salud a nivel de panel. Con sensores V/I/irradiancia/temperatura por módulo y publicación IoT (MQTT), [19] detectan degradaciones y activan alertas tempranas, lo que refuerza la utilidad del set mínimo de variables eléctricas+ambientales discutido en esta sección (Fig. 2.12).

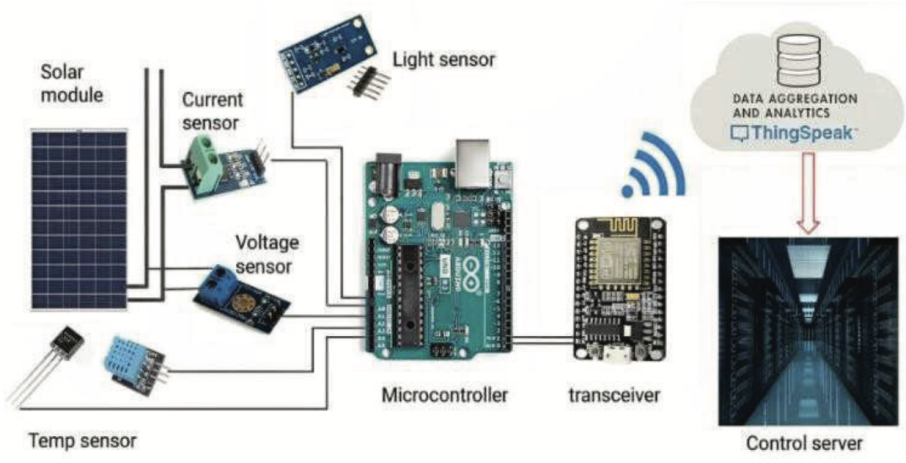


Figura 2.12: Arquitectura de monitoreo de salud por panel [19].

Bloques funcionales de un nodo económico (caso real). El diagrama de [13] resume la ruta sensado→preproceso→comunicación→nube en sistemas de bajo costo

con resultados de ML (RF $\sim$87 % para potencia).

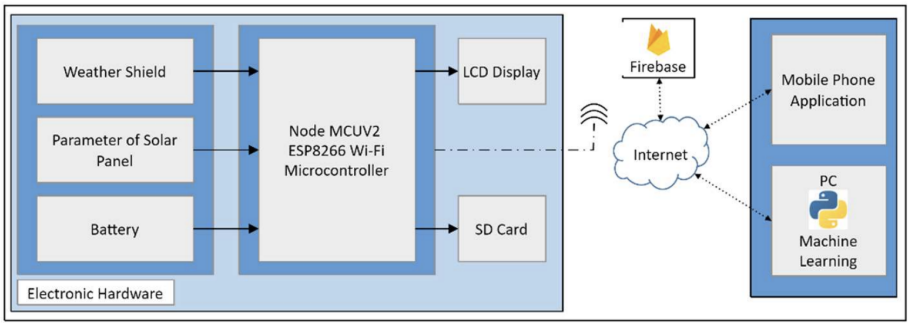


Figura 2.13: Bloques de un nodo IoT económico para FV [13].

2.5. Middleware y nube: persistencia, visualización y alertas

Marco teórico. El gateway/middleware recibe, valida y enriquece paquetes (RSSI/SNR, `ts_gw`), aplica *store-and-forward* con reintentos y publica a la nube. La capa de aplicación gestiona **series temporales**, **reglas** y **notificación** (correo, webhooks).

Evidencia complementaria. ThingSpeak (ThingSpeak) se usa en despliegues FV y casos académicos/industriales por su ingesta rápida, *dashboards* y reacciones (Alerts API, React/ThingHTTP, TalkBack) [11], [12], [21], [22]. El diseño de reglas con *persistencia* e *histéresis* reduce falsos positivos; además habilita **acción por evento** (p. ej., disparo de cámara) encadenando servicios ligeros.

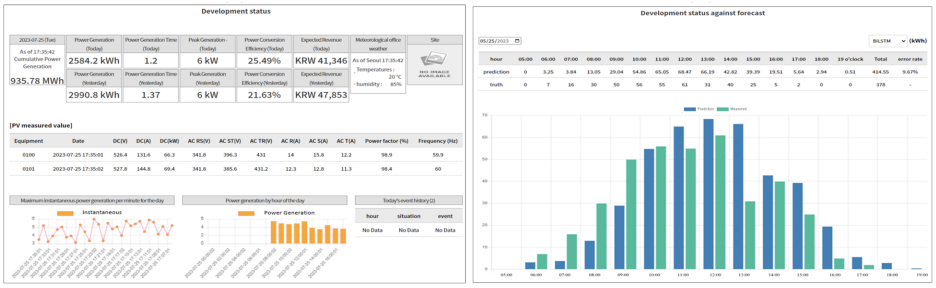


Figura 2.14: Ejemplo de *dashboard* multi-sitio con analítica/alertas [11].

Cuadro 2.4: Decisiones de diseño de aplicación y respaldo bibliográfico.

Decisión	**Justificación teórica**	**Respaldo**
Canales por nodo en ThingSpeak	Separación de responsabilidades; uso de API y visualización con gráficos dedicados	[21], [22]
Reglas con persistencia / histéresis	Disminución de falsos positivos en la generación de alertas	[11], [13]
Webhook / TalkBack para acciones	Orquestación ligera sin necesidad de exponer puertos locales	[12], [21]

Interfaces de usuario: tablero central y vista móvil. Además del tablero web multi-sitio [11], [13] muestran una app móvil que facilita operación en terreno (históricos, estado y alertas).



Figura 2.15: Ejemplo de interfaz móvil para monitoreo [13].

2.6. Verificación visual por evento: cámaras fijas y UAV

Marco teórico. La evidencia visual (RGB/térmica) confirma causas de caída de rendimiento: **sombras**, **suciedad**, **hot-spots**, **daños**. Puede ser *reactiva* (bajo alerta) o *periódica* (rondas).

Evidencia complementaria. UAV con cámaras RGB/IR han demostrado: navegación autónoma en pasillos de módulos (error $< 0.2\,\text{m}$ a $1.2\,\text{m/s}$) [23], detección de *hot-spots* con mapeo georreferenciado [24], y estimación de *soiling* a nivel celda (error RMS $< 1\,\%$) [25]. En nuestra arquitectura, la cámara fija (ESP32-CAM/RPi) se integra como **acción por evento** desde la capa de aplicación; UAV queda como extensión para inspecciones ampliadas [26].

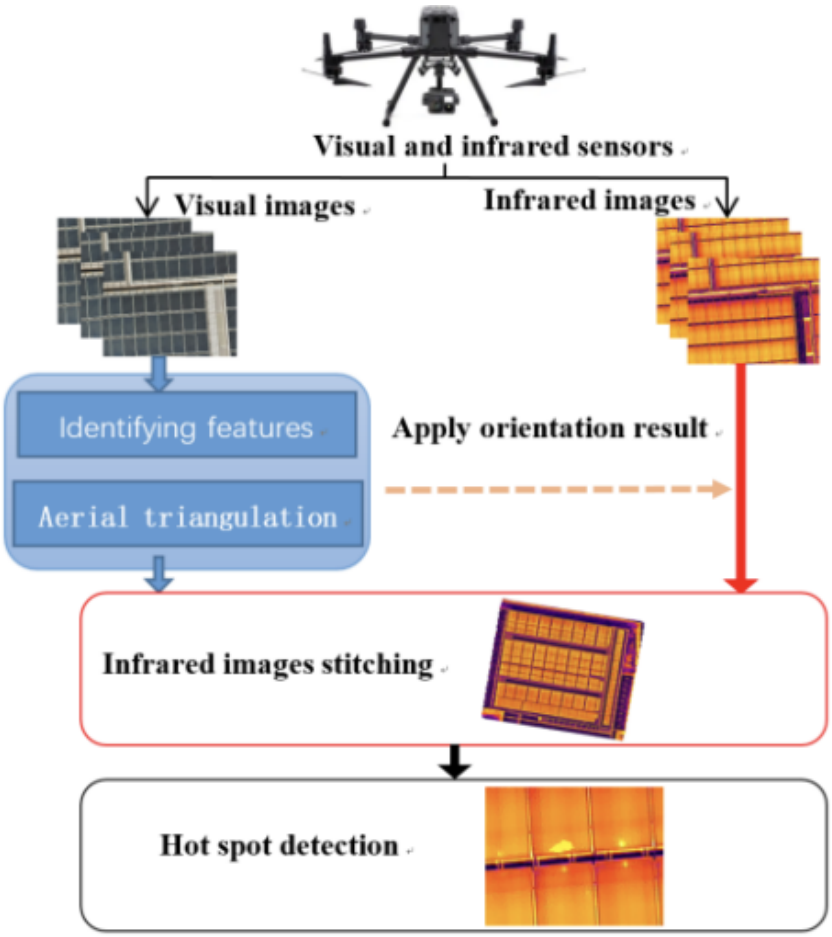


Figura 2.16: Inspección autónoma de filas con RGB/térmica [23].

2.7. Conclusiones del marco integrado

La literatura reciente respalda: (i) **arquitecturas por capas** con *edge* ligero para telemetría FV [11], [12]; (ii) uso de **LoRa/LoRaWAN** para baja tasa y gran alcance, con parámetros ajustados a entorno [10], [14]-[17]; (iii) **sensores digitales de bajo costo** suficientes para diagnóstico operativo a 30 min [13], [19]; (iv) **nube con dashboards y reglas** (ThingSpeak) para visualización/alertas [21], [22]; y (v) **verificación visual por evento** (cámara/UAV) que mejora interpretación y priorización de O&M [23]-[26]. Estas evidencias fundamentan las decisiones de diseño adoptadas.

2.8. Síntesis comparativa

La Tabla 2.5 resume los trabajos estudiados y su pertinencia directa para esta memoria (incluye *todos* los papers utilizados).

Cuadro 2.5: Comparación de trabajos relacionados en monitoreo FV con IoT/LoRa y verificación visual (compilación completa).

Referencia	**Tecnologías/alcance**	**Hallazgos/Métricas clave**	**Lo que tomamos**
Utama et al. (2023) [11]	IoT multi-planta, *edge*, dashboards	BiLSTM (R^2=0,9664, MAPE 0,1982), Isolation Forest (95.17 %); operación multi-sitio	Patrón por capas con *edge*, dashboards y analítica
Lee et al. (2023) [12]	EMS 5 capas; híbrido on-premise/nube	Continuidad operativa multi-sitio; comparación ZigBee/WiFi/Celular/NB-IoT/LoRa	Separación de responsabilidades; redundancia y selección de enlace
Demir et al. (2023) [13]	Nodo IoT bajo costo + ML	RF ∼87 % (potencia); flujo completo sensado→nube	Buenas prácticas sensado/JSON/visualización
Paredes-Parra et al. (2019) [16]	LoRa en planta FV	Coberturas $>$ 1 km con bajo consumo; telemetría periódica robusta	Topología nodo–gateway–nube (LoRa)
Health (2021) [19]	IoT (WiFi+MQTT) por panel	Alertas tempranas por degradación y condiciones ambientales	Variables eléctricas+ambientales mínimas útiles
Qian et al. (2022) [15]	LoRaWAN (SF/BW/CR)	PRR $>$90 % en urbano/rural; bajo consumo	Guía de parámetros LoRaWAN/robustez
Hameed & Kurnaz (2024) [18]	LoRaWAN + IA en el borde	SVM $>$90 % en nodo; cifrado end-to-end	Seguridad y factibilidad de *edge analytics*
Tradacete-Ágreda (2024) [20]	IoT a nivel de panel (I–V/EL)	Reconfiguración activa; diagnóstico fino por celda	Inspiración para acciones locales por evento
Kim et al. (2022) [17]	LoRa P2P en *junction box*	Detección térmica en diodos; enlace robusto con obstáculos	Opción P2P simple para pilotos
Benavides et al. (2023) [22]	ThingSpeak (ThingSpeak)	Canales PV/tiempo; reglas y visualización en tiempo real	Patrón de canales/umbrales y API
Segovia Ramírez et al. (2024) [26]	UAV + CNN + IoT	Flujo visual por evento; detección de fallas	Justifica módulo de evidencia visual
Augustin et al. (2016) [10]	Fundamentos LoRa (CSS, SF/BW/CR)	Sensibilidad y *trade-offs* de configuración (alcance vs airtime/robustez)	Parámetros de enlace y límites físicos
Guo et al. (2021) [27]	LoRa en campo (PRR/RSSI/SNR)	Factores que afectan PRR y robustez en distintos entornos	Expectativas realistas de enlace en sitio

Capítulo 3

Arquitectura del sistema

Este capítulo describe la arquitectura propuesta para el sistema IoT de monitoreo de paneles solares, con foco en validación a pequeña escala en viviendas, azoteas y granjas. La solución sigue el esquema por capas: **percepción**, **comunicaciones**, **middleware** y **aplicación**. Se presenta además la selección tecnológica y los criterios operativos que aseguran bajo consumo, trazabilidad y escalabilidad.

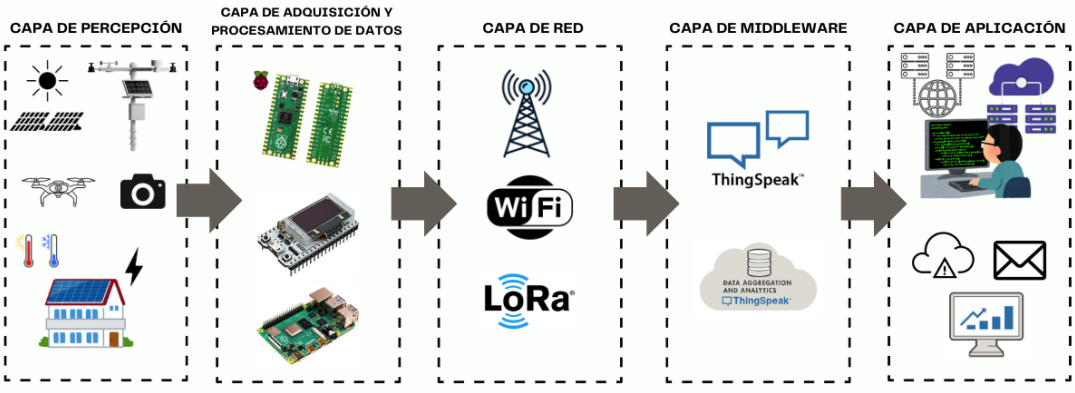


Figura 3.1: Topología de capas y tecnologías **adquisición–percepción** (NS), **red** (LoRa), ***middleware*** (gateway/servicios) y **aplicación** (nube).

3.1. Capa de percepción

La capa de percepción transforma magnitudes físicas en datos útiles para el resto del sistema. En este proyecto mide: temperatura y humedad ambiente, iluminancia (proxy de irradiancia), y parámetros eléctricos del módulo FV (tensión y corriente). En el *edge* se limita a tareas ligeras: sello temporal (`ts`), validaciones de plausibilidad, y **empaquetado JSON**. El cómputo agregado (promedios, percentiles) se realiza fuera de esta capa.

Objetivos funcionales

- Adquirir T/RH, lux, V_{pv}, I_{pv} con precisión suficiente para tendencias y detección de anomalías.
- Aplicar validaciones locales (*quality gate*) independientes del clima y marcar `qf`.
- Timestampear y empaquetar en JSON compacto para su transporte por LoRa.
- Exponer líneas de salida para actuación local (p. ej., disparo de cámara) frente a condiciones definidas.

Componentes

Nodo y radio

- **Raspberry Pi Pico W** (MCU principal): sensado, validaciones, empaquetado JSON.
- **Heltec TX LoRa** (módulo transmisor): recibe JSON por UART desde la Pico y lo emite por LoRa. Alimentado desde la Pico.

Sensores

- **T/H ambiente:** DHT22 (protocolo de una sola línea digital).
- **Iluminancia:** VEML7700 por I^2C (ajuste de ganancia/tiempo de integración para evitar saturación).
- **Eléctricos FV:** INA260 por I^2C (tensión, corriente; potencia nativa disponible). La Pico puede verificar $P_{vi} = V_{pv} \cdot I_{pv}$.

Actuadores

- **Disparo de cámara:** línea digital para activar una cámara USB (en SBC) o un dron (vía tarea en capa superior), útil al evidenciar anomalías.
- **Indicadores locales:** Pantalla/LED para señalización básica de eventos (alerta de umbral, error de sensor).

Integración eléctrica y buses

Los dispositivos I^2C (VEML7700, INA260) comparten SDA/SCL con *pull-ups* de 4,7 kΩ a 3,3 V. El DHT22 utiliza una línea digital dedicada con resistencia de *pull-up* según hoja de datos. El INA260 se ubica en serie (Vin + / Vin -) en el lazo del panel.

Cuadro 3.1: Conexiones entre la Raspberry Pi Pico W y los sensores

Componente	Alimentación	Líneas de comunicación
VEML7700 (iluminancia)	3.3 V / GND	SDA, SCL (I^2C)
DHT22 (T/H)	3.3 V / GND	GPIO (una sola línea + pull-up)
INA260 (V/I)	3.3 V / GND	SDA, SCL (I^2C)
Raspberry Pi Pico W	5 V (USB/VSYS)	I^2C (VEML/INA) y GPIO (DHT22)

La Figura 3.2 muestra el esquema de conexión física entre la Pico W y los sensores: VEML7700 (irradiancia), DHT22 (temperatura y humedad), e INA260 (corriente y voltaje). Los tres comparten líneas SDA y SCL, con alimentación a 3.3 V. El

INA260, además, se conecta en serie con el panel solar mediante sus terminales Vin+ y Vin- para medir la corriente de salida.

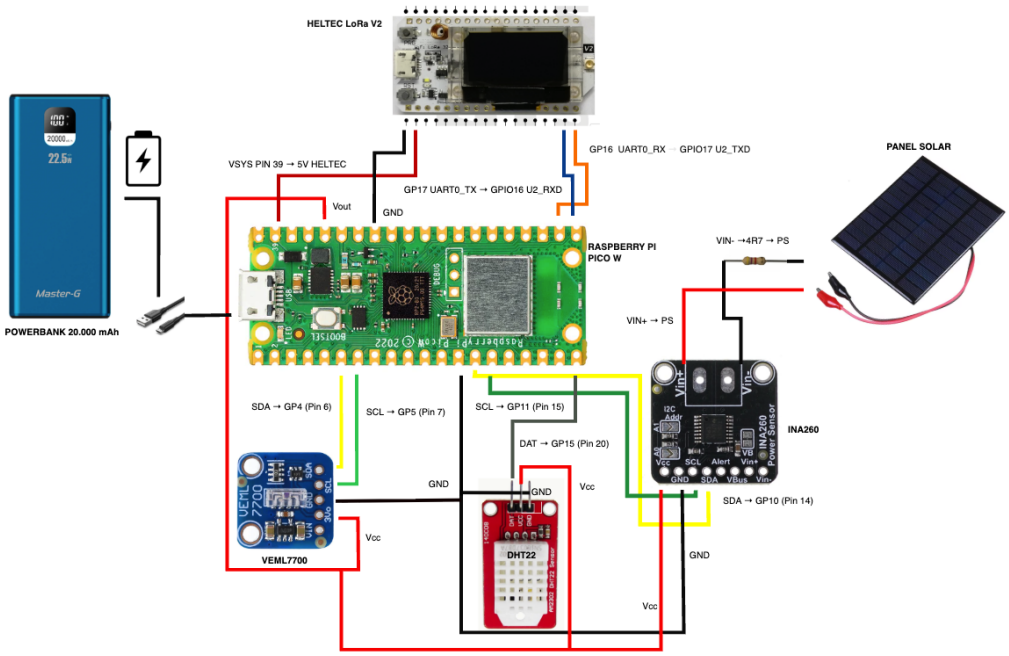


Figura 3.2: Conexión de los sensores VEML7700, DHT22 e INA260 a la Raspberry Pi Pico W.

Ubicación física y montaje

- **Irradiancia (VEML7700):** Se orienta con el mismo ángulo del panel y libre de sombras; idealmente en escenario real proteger con difusor claro y caja ventilada IP65.

- **T/H ambiente:** Idealmente se debe alojar en caja ventilada (radiación indirecta) para evitar sobrelectura por insolación directa.

- **Medición eléctrica:** instalar cercano a salidas del panel, minimizando lazo de corriente y asegurando conexiones firmes.

Muestreo y preprocesamiento local

En **operación**, el nodo toma **una medición puntual cada 30 min** y la publica; no hay muestreo continuo entre marcas. En **ensayos de laboratorio**, se usa muestreo a segundos ($T_s \in [1, 5]$ s) para calibración/validación; estas series pueden remuestrearse a 30 min para comparación con la operación.

La elección de una cadencia de 30 min en operación se fundamenta en que, las variables ambientales (T, RH) presentan variaciones lentas a escala horaria y la irradiancia cambia de forma moderada intra–hora (p. ej., paso de nubes). Esta decisión reduce consumo energético y tráfico de datos sin perder las tendencias diarias relevantes para diagnóstico FV.

Cada medición se empaqueta en un mensaje JSON compacto con la estructura mínima. La codificación y el reenvío ocurren posteriormente en la capa superior a través de los heltec.

Criterios de calidad y calibración

- **Plausibilidad independiente del clima.** Se usan límites físicos y de hoja de datos:
  - $V_{pv} \geq 0$, $I_{pv} \geq 0$, $P_{pv} \geq 0$ y $P_{pv} \leq V_{pv}^{\text{máx}} \cdot I_{pv}^{\text{máx}}$ (según especificaciones del sensor/panel).
  - *Detección de "stuck sensor"*: mismo valor exacto durante M marcas consecutivas ($M \geq 3$) $\Rightarrow$ `qf=0` salvo en *modo noche* (lux ≈ 0).
  - *Cotas de variación entre marcas*: $|\Delta x| \leq r_x^{\text{máx}}$ por cada 30 min (p. ej., para T y RH), fijadas por dinámica razonable del sensor, no por el clima.
- **Consistencia de potencia. (Opcional)** Comparación entre potencia nativa del INA260 y potencia calculada en la Pico:

$$\frac{\left| P_{\text{pv}}^{\text{INA260}} - (V_{pv} \cdot I_{pv}) \right|}{\text{máx}\left(P_{\text{pv}}^{\text{INA260}},\, \varepsilon_{\text{mín}}\right)} \leq \varepsilon,$$

con tolerancia relativa ε (típ. 5 %) y umbral absoluto $\varepsilon_{\text{mín}}$ para evitar divisiones por valores muy bajos. Si la discrepancia supera ε en K marcas consecutivas ($K \geq 2$), `qf=0`. En *modo noche* (lux ≈ 0) esta verificación puede omitirse.

- **Integración y ajuste de sensores.**
  - *VEML7700 (lux).*
    1. *Parámetros:* seleccionar ganancia/tiempo de integración para evitar saturación en máximos y mantener SNR en mínimos; verificar flags de saturación.
  - *INA260 (V, I, P).*
    1. *Linealidad:* barrer con cargas conocidas para obtener {I,V,P}; comparar con instrumento de referencia y reportar error relativo por tramos de corriente.
    2. *Montaje:* conexiones firmes, baja resistencia parásita en el lazo (Vin+/Vin−); verificar que el rango de tensión/corriente del sensor cubre el del panel de prueba.
  - *Temperatura/Humedad (DHT22).*
    1. *Montaje:* protección contra radiación directa (radiation shield) y buena ventilación para evitar sesgos térmicos.
    2. *Calibración:* ajuste con instrumento de referencia en condiciones estables (1–2 puntos) para compensar *offset* (y pendiente si procede).
- **Integridad de datos y de trama.**
  - *Esquema JSON:* validar presencia/tipo de campos obligatorios (`ts`, `node_id`, T, RH, lux, V_{pv}, I_{pv}, P_{pv}, `qf`).
  - *Coherencia temporal:* marca `ts` monótona y razonable respecto al reloj del *gateway*.

- *Checksum simple:* incluir `sum/crc` sobre campos numéricos para detectar corrupción de *payload*.
- *Política `qf`:* asignar `qf=1` sólo si superan plausibilidad, consistencia de potencia, esquema válido y `ts` coherente.

Gestión energética

Entre marcas, la Pico entra en *bajo consumo* (reloj/temporizador activos), despertando para sensar, empaquetar y transmitir. La Raspberry Pi Pico W soporta estados de*idle/light sleep* que reducen consumo sin perder estado del programa. Mantener sensores en *standby* cuando aplique y minimizar la ventana activa de radio. Con cadencia 30 min y *payload* < 120 B, el *duty cycle* es mínimo.

Este enfoque minimiza el *duty cycle* efectivo de CPU y radio sin interrumpir el sistema, preservando el régimen operativo (30 min) y extendiendo la autonomía con la batería disponible.

3.2. Capa de comunicaciones

La capa de comunicaciones garantiza el transporte confiable y eficiente de los datos desde la capa de percepción hasta el middleware y la nube. Se adopta una **ruta única** de transporte: **LoRa punto-a-punto (915 MHz)** desde el nodo hasta el *gateway*. El **nodo** está formado por la **Raspberry Pi Pico W** (sensado, validación ligera y **empaquetado JSON**) y un **Heltec transmisor (TX LoRa)** que recibe el JSON codificado por **UART** y lo *encapsula* en una trama LoRa **sin modificar el contenido de aplicación**. El **gateway Heltec (RX LoRa)** recibe la trama, decodifica y sube el registro a la nube (ThingSpeak) usando su propio enlace a internet. El nodo **no** publica a internet.

Objetivos funcionales

- Mantener la cadencia operativa de **30 min por medición** con tiempo en aire mínimo.

- Asegurar **robustez** (CRC de enlace, deduplicación, reintentos razonables) sin sobrecargar el nodo.

- Mantener **interoperabilidad** y trazabilidad temporal mediante un *payload* JSON **generado sólo en la Pico**.

Arquitectura y flujo de enlace

- **Nodo (Pico W + Heltec TX LoRa).**

  - *Pico W:* toma una medición **cada 30 min**, valida y **empaqueta** un JSON con `ts`, `node_id`, T, RH, lux, V_{pv}, I_{pv}, P, `qf`.
  - *Enlace interno* Pico→Heltec: **UART TTL**.Pines: TX/RX y **GND común**.
  - *Heltec TX LoRa:* recibe el JSON por UART, codifica el paquete y lo transmite por LoRa. **Alimentación:** el Heltec se alimenta desde la Pico (5 V hacia su regulador.

- **Gateway Heltec (RX LoRa → nube).**

  - Recibe la trama LoRa (CRC de enlace habilitado), **extrae el JSON tal cual y decodifica**, añade metadatos (`ts_gw`, RSSI/SNR, `ingress=lora`, `schema`) y publica a ThingSpeak.
  - Opera *store-and-forward* cuando no hay internet y aplica deduplicación por {`node_id`, `ts`} (detalles en la capa de middleware).

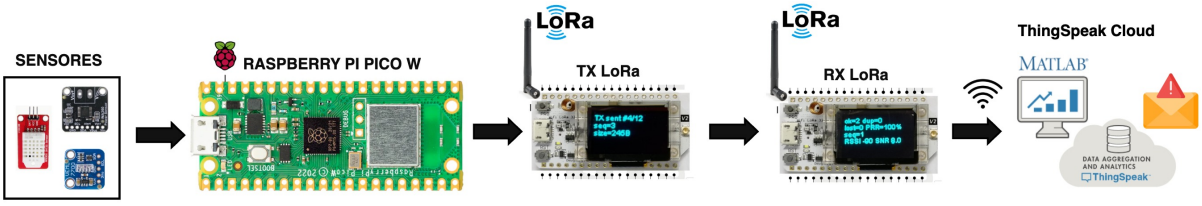


Figura 3.3: Flujo: Sensores → Pico (JSON) → UART → Heltec TX (LoRa) → Heltec RX (gateway) → nube (ThingSpeak).

Parámetros de enlace LoRa (propuestos)

Cuadro 3.2: Parámetros de enlace LoRa

Banda	ISM 915 MHz
Modulación	LoRa (CSS)
Ancho de banda (BW)	125 kHz
Spreading Factor (SF)	9 (ajustable 7–12 según RSSI/SNR)
Coding Rate (CR)	4/5
Potencia TX	17–20 dBm (según normativa/antena)
Preámbulo	8 símbolos; **CRC** habilitado
Payload típico	80–120 bytes (JSON compacto)

Airtime de referencia: a BW=125 kHz, SF=9, CR=4/5 y 100 B, el *airtime* es ∼0,55 s.

Con una marca cada 30 min, la ocupación del canal es despreciable.

Formato de paquete y canal

- **Aplicación (generada en la Pico).** JSON compacto:

```
{ "schema":1, "ts":<epoch_ms>, "node_id":"<hex/id>",
  "mode":"op|lab", "t_amb_C":<f>, "rh_pct":<f>, "lux":<f>,
  "v_pv_V":<f>, "i_pv_A":<f>, "p_pv_W":<f>, "qf":<0|1> }
```

(campos opcionales: `p_vi_W`, `night_mode`).

- **Enlace interno Pico→Heltec TX (UART).** Usa el salto de linea para separar (`\n`) o los caracteres especiales STX/ETX, y **longitud** o **COBS** opcional para robustez. (COBS: Consistent Overhead Byte Stuffing) *Handshake* simple (p. ej., `OK`/`ERR`) opcional.

- **Transporte LoRa (Heltec TX).** La librería de radio añade CRC (Cyclic Redundancy Check) de enlace; **no se altera** el JSON de aplicación.

- **Publicación en gateway (Heltec RX).** Preserva el JSON, añade metadatos (`ts_gw`, RSSI/SNR, `ingress`), ejecuta deduplicación y publica a la nube.

Fiabilidad y reintentos

- **CRC de enlace:** activo en LoRa; el gateway descarta tramas con error.

- **ACK opcional:** si el Heltec TX abre una ventana RX breve tras TX, el gateway puede responder un *ACK* mínimo con el `ts` o `seq`. Si no se implementa ACK, se usa *reenvío diferido*: la Pico re-envía la última marca *piggyback* en el siguiente periodo si el gateway reporta ausencia.

- **Store-and-forward en gateway:** ante caída de internet, cola local con *backoff* y TTL por paquete.

- **Deduplicación:** clave $\{$`node_id`, `ts`$\}$ en gateway/nube.

Temporización y sincronización

- **Nodo:** cron interno estrictamente cada 30 min; la Pico sólo transmite cuando ha generado un JSON válido.

- **Gateway.** Sincronizado por NTP; añade `ts_gw`. Si $|\texttt{ts} - \texttt{ts\_gw}|$ excede umbral, adjunta `drift_ms` para que en capa de (middleware/app) puedan detectar y corregir la desincronización de reloj.

Cumplimiento y operación

- **Espectro:** operación en ISM 915 MHz cumpliendo potencia/ocupación local.
- **Gestión de SF/BW:** SF9 inicial; ajuste manual (o guiado por RSSI/SNR del gateway) entre 7 y 12 para equilibrar sensibilidad y *airtime*.
- **Cadencia y tamaño: 1 paquete/30 min**; mantener *payload* < 120 B para holgura si sube SF o se añaden metadatos.

Resumen operativo

1. Pico: sensa $\rightarrow$ valida $\rightarrow$ **empaqueta JSON**.
2. UART: Pico $\rightarrow$ Heltec TX (framing simple).
3. Radio: Heltec TX **transmite LoRa** (CRC activo).
4. Gateway: Heltec RX recibe $\rightarrow$ añade metadatos $\rightarrow$ publica a ThingSpeak; *store-and-forward* si falta internet.

3.3. Capa de middleware

La capa de *middleware* se implementa en el **gateway Heltec (RX LoRa)** y opera como puente confiable entre el enlace de radio y los servicios de aplicación. Su responsabilidad es **ingerir** tramas LoRa, **extraer y validar** el JSON generado por la Pico, **enriquecer** con metadatos operativos, **encolar y reenviar** ante fallos, y **publicar** de forma idempotente a la nube (ThingSpeak). El nodo no accede a internet.

Objetivos y alcance

- Garantizar la entrega de cada marca de 30 min con trazabilidad temporal y de enlace.

- Aislar claves, endpoints y políticas de publicación del firmware del nodo.
- Proveer mecanismos de calidad de servicio: validación de esquema, deduplicación, *store-and-forward* y control de reintentos.

Flujo funcional (extremo a extremo)

1. **Ingesta LoRa**: recepción y verificación de CRC a nivel de radio.
2. **Extracción**: decodificación de la carga útil y obtención del JSON .
3. **Validación de esquema**: presencia y tipos de campos obligatorios (`ts`, `node_id`, `t_amb_C`, `rh_pct`, `lux`, `v_pv_V`, `i_pv_A`, `p_vi_W`, `qf`).
4. **Enriquecimiento**: añadido de metadatos del gateway y de enlace.
5. **Entrega**: publicación a ThingSpeak con control de idempotencia; si falla, **cola local** y reintentos.

Validación de mensajes

La validación es **sintáctica y de coherencia mínima**; las reglas físicas y de calidad ya fueron aplicadas en el nodo (vía `qf`).

- **Esquema y tipos**: campos obligatorios presentes y con tipos numéricos válidos; rechazo de NaN/Inf.
- **Coherencia temporal**: el gateway está sincronizado por NTP y añade `ts_gw`. Si $|\texttt{ts} - \texttt{ts\_gw}| > \tau$ (umbral), adjunta `drift_ms` para diagnóstico.

Enriquecimiento y versionado

El middleware **no altera** el contenido de aplicación; sólo agrega metadatos y controla el versionado del *payload*.

```
{
  "schema": 1,                  // versión del payload
  "ts_gw": <epoch_ms>,          // marca de tiempo del gateway (NTP)
  "rssi": <int>, "snr": <float>,   // calidad de enlace LoRa
  "ingress": "lora",            // origen
  "gw_fw": "<vX.Y>",            // versión de firmware del gateway
  ... <payload original de la Pico> ...
}
```

Política de versión: aumentos menores (`schema` 1→1.x) mantienen compatibilidad hacia atrás; aumentos mayores requieren mapeo explícito antes de publicar.

Cola, reintentos e idempotencia (*store-and-forward*)

Cuando no hay internet o la API rechaza la operación, el gateway **encola** el registro (RAM/flash) con:

- **Clave lógica**: {`node_id`, `ts`} para **deduplicación** e **idempotencia** aguas arriba.
- **Backoff exponencial**: 1–2–4–8 min (configurable) y **TTL** por paquete.
- **Orden FIFO**: preserva la serie temporal al vaciar la cola.

Publicación a ThingSpeak (REST/HTTP)

El gateway publica únicamente vía REST/HTTP usando la API oficial de ThingSpeak: `POST` con la *Write API Key* y el mapeo:

Field1=T, Field2=RH, Field3=lux, Field4=V_{pv}, Field5=I_{pv}, Field6=P_{pv}, Field7=`qf`. Esta vía evita mantener sesiones MQTT y es suficiente para la cadencia de 30 min.

Servicios de integración (alertas/acciones)

Sin añadir orquestadores externos, el middleware soporta dos patrones para ejecutar acciones desde ThingSpeak, manteniendo la ruta Pico→LoRa→gateway→ThingSpeak:

- **Webhook (React + ThingHTTP → gateway).** Cuando la regla se cumple, *React* invoca *ThingHTTP*, que realiza un `POST` al endpoint del gateway (p. ej., `/camera/snap`). *Requisitos:* el endpoint debe ser accesible desde internet (exposición controlada, redirección de puertos o túnel seguro). *Contrato:* solicitud con token (cabecera `Authorization: Bearer <token>`) y respuesta JSON con `status` e `url` de la imagen.

- **Cola de comandos (TalkBack ↔ gateway).** Alternativa robusta tras NAT (sin puertos abiertos). La regla escribe un comando (p. ej., `SNAP` con parámetros opcionales) en *TalkBack*; el gateway consulta (*polling*) a intervalos regulares, ejecuta la acción (captura) y publica la URL (en `status` o canal asociado).

Las alertas por correo se implementan con *Alerts API* y se describen en detalle en la Capa de aplicación.

Seguridad y gestión de secretos

- **Resguardo de claves.** Las *API keys* (ThingSpeak, TalkBack, Alerts) residen sólo en el gateway (variables de entorno/archivo seguro). Principio de privilegio mínimo y rotación periódica.

- **Transporte seguro.** Publicación a ThingSpeak exclusivamente por **REST/HTTPs**. Para webhooks, exigir TLS y token de aplicación (cabecera `Authorization`) y validar origen.

- **Integridad del payload (opcional).** Si el nodo firma/cifra, el gateway verifica HMAC o descifra AEAD (AES-GCM) antes de publicar. Gestión de *nonce*/`ts` para evitar repeticiones.

- **Endurecimiento operativo.** Límite de tamaño de *payload*, validación estricta de JSON, *rate limiting* y *backoff* exponencial en reintentos para mitigar abusos de red o caídas de API. Nunca registrar secretos en *logs*.

Observabilidad y operación

- **Métricas.** Latencia extremo a extremo (nodo→nube), tasa de entrega, tamaño medio de *payload*, profundidad de cola (*store-and-forward*), % de lecturas con `qf=1`, tasa de duplicados, RSSI/SNR.

- **Registro.** Eventos de validación fallida, deduplicación, caídas/reconexiones del enlace externo, activaciones de servicios (`camera snap`, TalkBack) y errores de publicación (códigos HTTP).

- **Supervisión.** *Heartbeat* del gateway; alerta si no se reciben datos de un nodo durante N intervalos de 30 min. Umbral de `drift_ms` para detectar desincronización temporal.

3.4. Capa de aplicación

La capa de aplicación entrega **visualización**, **diagnóstico** y **automatización de alertas/acciones**. Se utiliza ThingSpeak para almacenamiento, paneles y servicios de reacción (MATLAB Analysis, TimeControl, Alerts API, React/ThingHTTP y TalkBack). Toda la lógica asume una **cadencia de 1 medición cada 30 min**.

Objetivos

- Mostrar series temporales graficamente (P_{pv}, V_{pv}, I_{pv}, lux, T/RH) con metadatos (RSSI/SNR y `qf`).

- Detectar **anomalías multivariantes** con **persistencia e histéresis**, notificar por

correo y **asociar una hipótesis de falla** (sombreado/suciedad, desconexión, fallo eléctrico, perdida de angulo etc).

- Disparar **captura de imagen** como evidencia del evento y publicar la URL junto al registro.

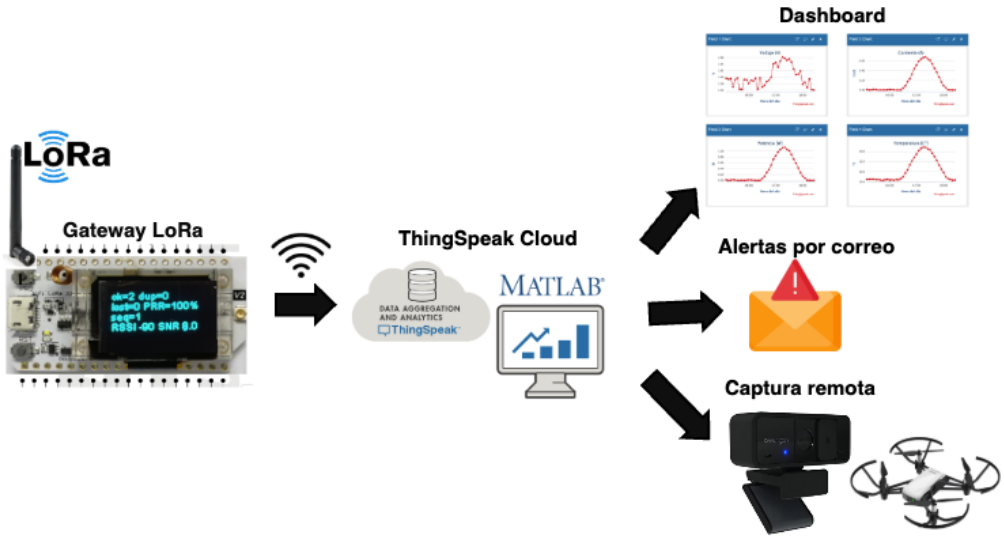


Figura 3.4: Capa de aplicación: ThingSpeak recibe datos del gateway LoRa y entrega tres funciones principales: visualización en paneles, alertas por correo y disparo de captura remota (webcam o dron).

Ingesta, lectura y persistencia

El gateway publica por **REST/HTTP** (`POST /update.json`) al canal de ThingSpeak. La aplicación:

1. Visualiza *widgets* de serie temporal y estado del enlace.
2. Lee datos con `thingSpeakRead` desde *MATLAB Analysis* (gatillado por *TimeControl* cada 30 min) usando la *Read API Key* del canal.

Se recomienda conservar `ts_gw`, RSSI y SNR en `status` o un campo dedicado para auditoría.

Paneles y gráficos (síntesis operativa)

Se mantienen cuatro vistas mínimas orientadas a operación; cada una aporta un indicador claro de salud del sistema o del enlace sin entrar en análisis avanzado.

Cuadro 3.3: Vistas del panel, propósito e indicadores de problema

Vista	**Propósito**	**Señales de alerta**
Serie 24–72 h: P_{pv}, V_{pv}, I_{pv}, lux, % `qf=1`	Tendencia diaria y continuidad de datos (marcas nocturnas con lux≈ 0)	Vacíos de datos, `qf` bajo sostenido, serrucho atípico en P_{pv}
Correlación diaria P_{pv}–lux	Coherencia producción–luz (sombreado/suciedad)	Pendiente/ganancia reducida vs. días sanos
Perfil diurno normalizado $P_{pv}/P_{pv}^{\text{máx}}$	Comparación entre días; simetría matutina/vespertina	Asimetrías persistentes (sombras por orientación/obstáculos)
Salud del enlace: RSSI/SNR y tasa de entrega	Calidad de radio y entrega extremo a extremo	RSSI/SNR degradados, entregas <100 %, necesidad de ajustar SF/BW

Motor de alertas (MATLAB Analysis + Alerts API)

El bloque de *MATLAB Analysis* lee la **última marca** del canal (p. ej., `Field1=I`, `Field2=V`, `Field3=lux`, `Field4=T`, `Field5=RH`) y calcula $P_{pv} = V \cdot I$. Con umbrales configurables, evalúa reglas **multivariantes** y, si corresponde, envía correo vía **Alerts API** (cabecera `ThingSpeak-Alerts-API-Key`). Los secretos se mantienen como `<READ_KEY>` y `<ALERTS_KEY>`.

Umbrales recomendados (ajustables).

`HIGH_IRRADIANCE`, `LOW_IRRADIANCE`, `HIGH_TEMP`, `LOW_TEMP`, `HIGH_HUMIDITY`,

`LOW_VOLTAGE`, `HIGH_VOLTAGE`, `LOW_CURRENT`, `HIGH_CURRENT`, T_{hold}, H_P.

Reglas implementadas (ejemplos operativos). Se expresa el patrón (**C**orriente, **V**oltaje, **L**ux, **T**emperatura, **H**umedad) y la hipótesis resultante; la alerta se emite sólo si persiste T_{hold} (múltiplos de 30 min) y se limpia por histéresis (H_P):

Cuadro 3.4: Patrones multivariantes → diagnóstico probable → acción

Condición (C;V;L;T;H)	**Hipótesis**	**Acción**
`lux` > `HIGH_IRRADIANCE` & V < `LOW_VOLTAGE` & I < `LOW_CURRENT`	Alta luz + baja potencia ⇒ suciedad/sombreado/células dañadas	Correo + **foto**
I > `HIGH_CURRENT` & V < `LOW_VOLTAGE`	Corriente alta + voltaje bajo ⇒ cortocircuito parcial / batería descargada	Correo + **foto**
I < `LOW_CURRENT` & `lux` > `HIGH_IRRADIANCE`	Corriente nula con buena luz ⇒ fusible quemado / fallo de inversor	Correo + **foto**
V > `HIGH_VOLTAGE` & I < `LOW_CURRENT`	Voltaje alto + corriente baja ⇒ carga desconectada / batería cargada	Correo
V < `LOW_VOLTAGE` & `lux` > `HIGH_IRRADIANCE`	Voltaje muy bajo con buena luz ⇒ conexión deficiente / microfisuras	Correo + **foto**
H > `HIGH_HUMIDITY` & (baja V o alta I anómala)	Alta humedad + fluctuaciones ⇒ aislamiento deficiente / humedad en conectores	Correo
`lux` < `LOW_IRRADIANCE` & I > `HIGH_CURRENT`	Baja irradiancia + alta corriente ⇒ error de sensado / fuga de corriente	Correo (técnica)
`lux` < `LOW_IRRADIANCE` & T > `HIGH_TEMP`	Poca luz + alta temperatura ⇒ acumulación térmica o reflejos puntuales	Correo (técnica)

Limpieza por histéresis: una vez activa una regla, se despeja sólo si $P_{pv} > P_{\text{mín}} + H_P$

durante una ventana de confirmación. Las marcas con *modo noche* (lux≈ 0) inhiben reglas dependientes de luz.

Correo de alerta. El *MATLAB Analysis* prepara `subject` y `body` (listado de alarmas detectadas) y las envía vía **Alerts API** (`webwrite` al endpoint de alerts con cabecera `ThingSpeak-Alerts-API-Key`). El mensaje incluye `node_id`, `ts`/`ts_gw`, V, I, P, lux y, si aplica, `drift_ms` y enlace al canal.

Disparo de imagen y publicación de evidencia

El correo puede acompañarse de evidencia visual:

- **Webhook (React + ThingHTTP $\rightarrow$ gateway):** la regla activa *ThingHTTP*, que hace `POST` a `/camera/snap` en el gateway (endpoint público o túnel seguro). El gateway coordina una **Raspberry Pi + Camera/USB** (laboratorio), sube el JPEG a un repositorio y devuelve una **URL**.

- **TalkBack (cola de comandos):** alternativa tras NAT. La regla escribe `SNAP`; el gateway *pulsa* la cola, captura y publica la **URL**.

La URL se anexa al `status` (o a un campo dedicado) del mismo canal, vinculada al `ts` del evento.

Parámetros de aplicación

Cuadro 3.5: Parámetros y recomendaciones

$P_{\text{mín}}$, $\text{lux}_{\text{mín}}$	Umbrales base para evaluar la regla de potencia con luz suficiente
T_{hold}, H_P	Persistencia e histéresis para evitar falsos positivos
`HIGH/LOW_VOLTAGE`, `HIGH/LOW_CURRENT`	Límites eléctricos coherentes con el módulo/INA260
`HIGH/LOW_IRRADIANCE`, `HIGH_TEMP`, `HIGH_HUMIDITY`	Límites ambientales, ajustados por emplazamiento
N_{miss}	Intervalos sin datos para alertar caída de nodo/gateway
$\mathcal{R}_{\text{mail}}$	Lista de destinatarios de correo

Retención, exportación y auditoría

- **Retención:** marcas de 30 min en ThingSpeak; respaldo periódico (CSV/JSON).
- **Exportación:** descargas para análisis externo (MATLAB/Grafana).
- **Auditoría:** cada alerta/acción registra {`node_id`, `ts`, regla, estado, `url` (si aplica)}.

Resumen operativo

1. Gateway publica cada 30 min por REST/HTTP; paneles muestran variables y metadatos.

2. *TimeControl* ejecuta *MATLAB Analysis* → lectura del último dato → evaluación de reglas.

3. Si hay evento persistente: **Alerts API** envía correo y se activa **foto** (ThingHTTP o TalkBack); la **URL** queda asociada al evento en el canal.

Capítulo 4

Experimentos, algoritmos y datos

Este capítulo valida la solución propuesta mediante una secuencia de pruebas que replican el flujo real de datos: **adquisición (Pico)**, **transporte (LoRa)**, **middleware (gateway)** y **aplicación (ThingSpeak/alertas)**. La validación se organiza por *etapas*, desde banco de pruebas hasta piloto en campo, y se mide con indicadores reproducibles.

4.1. Objetivos de la validación

- Verificar exactitud/estabilidad de sensores (T/RH, lux, V_{pv}, I_{pv}) y cálculo $P_{pv} = V \cdot I$ en el *edge*.
- Confirmar confiabilidad de enlace LoRa (entrega, *airtime*, robustez) a **1 marca/30 min**.
- Corroborar integridad extremo a extremo: JSON $\rightarrow$ gateway $\rightarrow$ ThingSpeak.
- Evaluar el **motor de alertas multivariante** (persistencia + histéresis) y el **disparo de imagen**.
- Caracterizar la **sensibilidad angular del módulo** (potencia P_{pv} vs. orientación 0°/45°/90°) con carga fija.

4.2. Algoritmos implementados (resumen operativo)

Nodo (Pico W) — sensado y empaquetado

- Cadencia: cada **2 seg**. Lecturas: T/RH (DHT22), lux (VEML7700), V, I (INA260).
- Cómputo local: $P{=}V{\cdot}I$ y bandera de calidad `qf` (plausibilidad y consistencia).
- Mensaje: JSON con {`schema`, `ts`, `node_id`, T, RH, lux, V, I, P, `qf`, `mode`}.
- Transporte: envío por LoRa al gateway.

Gateway — decodificación y publicación

- Verificación de integridad de enlace (CRC) y extracción de JSON.
- Enriquecimiento con metadatos {`ts_gw`, RSSI, SNR, `ingress`}.
- Publicación a ThingSpeak (REST/HTTP). Si no hay Internet: *store-and-forward* con *backoff*.

Aplicación — análisis y alertas

- Consulta de última marca (*TimeControl* cada 30 min).
- Evaluación de reglas multivariantes con **persistencia** e **histéresis**.
- En caso de evento: *Alerts API* (correo) y *ThingHTTP/TalkBack* para **captura de imagen**.
- Publicación de la URL de imagen en `status`/campo dedicado.

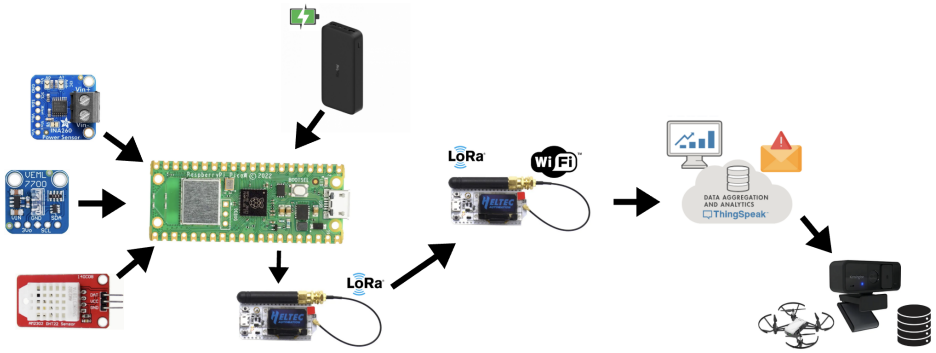


Figura 4.1: Flujo operativo extremo a extremo

4.3. Diseño experimental por etapas

Se validan cuatro etapas complementarias:

E1. Banco de sensores y firmware (escala de segundos).

E2. Enlace LoRa y consumo energético (laboratorio/campo controlado).

E3. Integración extremo a extremo (dato único/30 min a la nube).

E4. Piloto operativo y alertas/cámara (campo real, días consecutivos).

Cuadro 4.1: Trazabilidad objetivos ↔ etapas

Objetivo	**E1**	**E2**	**E3**	**E4**
Exactitud sensórica (T/RH, lux, V/I)	✓			✓
Cálculo $P{=}VI$ en nodo (*edge*)	✓		✓	✓
Confiabilidad LoRa (PDR, airtime)		✓	✓	✓
Integridad E2E JSON→nube			✓	✓
Alertas multivariantes + cámara				✓

Glosario de variables y banderas

Cuadro 4.2: Símbolos/campos usados en los experimentos.

Campo	**Descripción**	**Unidad**
`qf`	Bandera de calidad: 1=lectura válida (plausibilidad y consistencia superadas), 0=inválida.	–
`flags`	Banderas diagnósticas. `lux_sat`: sensor óptico saturado; `vbus_low`: bus de alimentación bajo.	–
V_{pv} (`V_V`)	Tensión del panel (medida por INA260).	V
I_{pv} (`I_A`)	Corriente del panel (INA260).	A
P_{pv} (`P_W`)	Potencia eléctrica del panel $P = V \cdot I$ (INA260).	W
`lux` / `lux_raw`	Iluminancia (VEML7700) / cuenta cruda (16 bit).	lux / –
T (`T_C`)	Temperatura de aire (DHT22).	°C
RH (`RH_pct`)	Humedad relativa (DHT22).	%
`R_load_ohm`	Resistencia de carga conectada al panel.	Ω
`ts` / `ts_gw`	Marca de tiempo en el nodo / en el gateway.	s (epoch)
`node_id`	Identificador del nodo.	–
`mode`	Etapa operativa (E1/E2/E3/E4).	–
`schema`	Versión del payload JSON.	–
`ina_present`	1 si INA260 está activo; 0 si no.	–
`Vbus_meas_V`	Tensión del bus de alimentación (medición auxiliar para diagnóstico).	V

Condiciones generales de experimentación. Todos los experimentos de caracterización con el panel fotovoltaico y los sensores ambientales se realizaron en **Valparaíso, Chile**, el **05 de septiembre de 2025**, en condiciones de cielo despejado y alta irradiancia solar. Se desarrollaron en horario diurno (aprox. **11:00–14:00 h**), lo que aseguró estabilidad de luz y una temperatura ambiente en torno a 27–28 °C. Si bien se ejecutaron pruebas en distintos días, para este documento se presentan únicamente los

registros correspondientes a las condiciones más estables y representativas.

Ensayo A1–A3: orientación del panel (0°/45°/90°)

Montaje y criterio. Para este experimento se utilizó un panel flexible USB (5 V) con **carga fija** R_{load}=4,7 Ω e INA260 activo (`ina_present=1`). La cadencia de muestreo fue de ∼2 s durante ∼4–5 min por ángulo. El ambiente presentó condiciones de alta luminosidad, con `lux_sat` en el VEML7700; todas las muestras registradas mostraron `qf=1`. Se **excluyen los primeros 20 s** de cada corrida, y se reporta la *mediana* de P_{pv}.



(a) 0° (b) 45° (c) 90°

Figura 4.2: Posiciones angulares del panel en los tres experimentos.

Observaciones de sensores El canal de luz indicó **saturación** del VEML7700 (“`lux_sat`”, `lux_raw`=65535), por lo que en estas corridas no discriminó variaciones angulares de iluminancia. Aun así, los **canales eléctricos** (INA260) y la bandera `qf=1` en todas las muestras sustentan el análisis de potencia. La **temperatura** y **humedad** se mantuvieron estables en el rango de 27 – 28 °C y 33 – 36 % RH, respectivamente, sin transitorios apreciables.

Cuadro 4.3: Potencia por orientación (mediana) y diferencia relativa vs. 90°.

Ángulo	P_{med} **[W]**	Δ **vs. 90°**	**Notas**
0°	0.341	$-52\,\%$	Serie estable; `qf=1`.
45°	0.701	$-1{,}3\,\%$	Rango estable $\sim$0.697–0.706 W; `qf=1`.
90°	0.710	—	Referencia; mayor P observado.

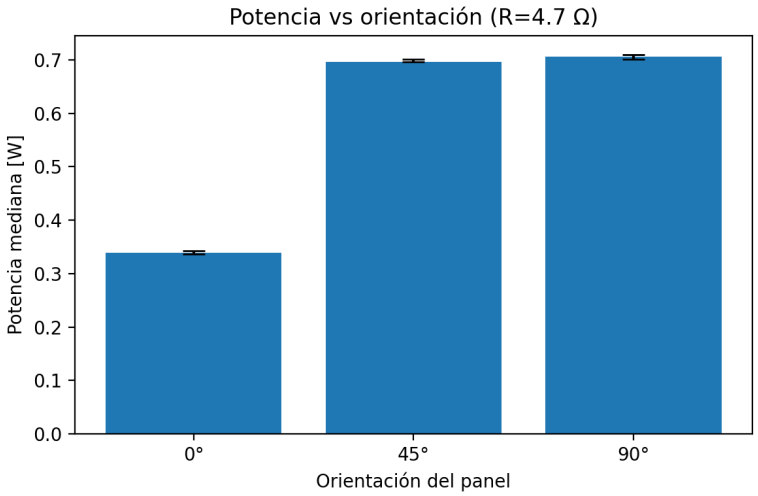


Figura 4.3: Comparación de potencia P_{pv} por orientación (0°/45°/90°). Se excluye el arranque (20 s).

Comentario y toma de decisiones. Con iluminancia alta (sensor óptico saturado, pero P *medida* con INA260), **90°** entrega la mayor potencia; **45° es muy cercano** ($\Delta \sim$1–1.5 %), mientras que **0°** es claramente inferior ($\sim$52 % menos que 90°). No se observaron lecturas inválidas (`qf=1` en todas); la diferencia no se debe a fallos. Operativamente, la penalidad de 45° frente a 90° es *baja*.

Ensayo A4-A5: sombras parciales y detritos (hojas y arena)

Montaje y criterio. Mismo panel y R_{load}=4,7 Ω con INA260 activo (modo E1). Se comparan: (i) **sombras parciales** en cuatro niveles (25 %, 50 %, 75 %, 100 %)

y (ii) **detritos**: *hojas* y *arena*. El canal óptico puede saturarse; la comparación se fundamenta en P, I y V.

Extracción de métricas. Para cada condición: potencia media $\overline{P}$, mediana $\tilde{P}$, desviación estándar σ_P y CV=$\sigma_P/\overline{P}$. En series transitorias, además: $\Delta P_{\text{máx}} = 100(1 - \text{mín}\, P/P_0)$, tiempos de caída/recuperación y déficit energético E_{def} respecto de P_0.

Cuadro 4.4: Sombras transitorias: resumen de métricas calculadas sobre los logs del experimento.

Escenario	$\Delta P_{\text{máx}}$ [%]	$t_{\downarrow}$ [s]	$t_{\uparrow}$ [s]	E_{def} [J]	E_{def} [%]
Lámina translúcida (50 %)	99.84	0	—	10.75	99.01
“Nube” (pasada lenta, 75 %)	100.00	0	—	13.54	100.00
“Nube” (pasada rápida)	99.92	0	56	65.46	99.97



(a) 25 % shading (b) 50 % shading (c) 75 % shading

Figura 4.4: Posiciones de sombreado parcial aplicadas al panel solar.



(a) Panel limpio (b) Panel con arena (c) Panel con hojas

Figura 4.5: Condiciones de limpieza y obstáculos aplicados al panel solar.

Comentario. En **sombras parciales**, el descenso de $\overline{P}$ es monótono con el porcentaje de oclusión: 25 % presenta la menor penalización y 100 % la mayor, con incremento del CV a medida que crece la oclusión. El caso **hojas** introduce irregularidad espacial y temporal: típicamente mayor CV que la arena y pequeños picos de recuperación mientras se mueve el material. La **arena** atenúa de forma más uniforme; su $\overline{P}$ se sitúa entre los niveles de sombra intermedia y 100 %, pero con CV menor que el de hojas. En transitorios, las hojas muestran rampas de caída/recuperación y un déficit energético E_{def} proporcional al tiempo de oclusión.

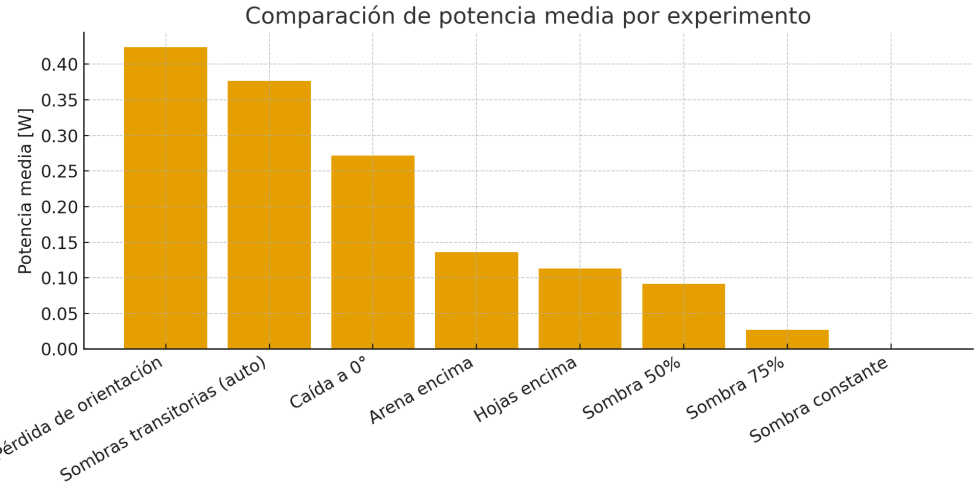


Figura 4.6: **Pérdida de orientación.** Serie temporal de potencia con P_0, t_{on}, $t_{\text{mín}}$ y t_{off} (si aplica). Caída máxima $\sim 42{,}5\,\%$ sin recuperación completa dentro del log.

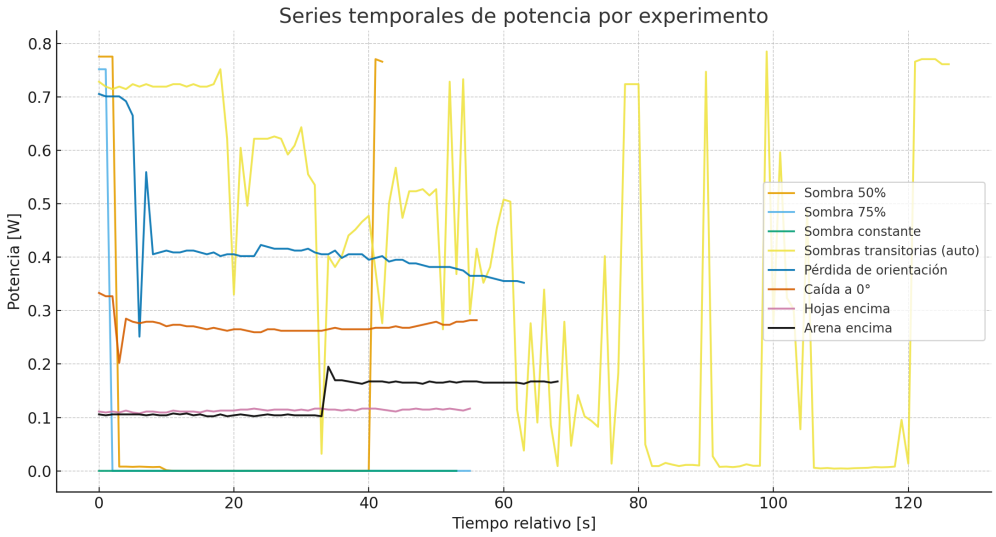


Figura 4.7: **Caída a** $0°$ **(1538).** Serie temporal de potencia. Mínimo a $\sim 6\,\mathrm{s}$ con caída $\sim 27{,}5\,\%$ y recuperación al $95\,\%\,P_0$ en el mismo instante medido.

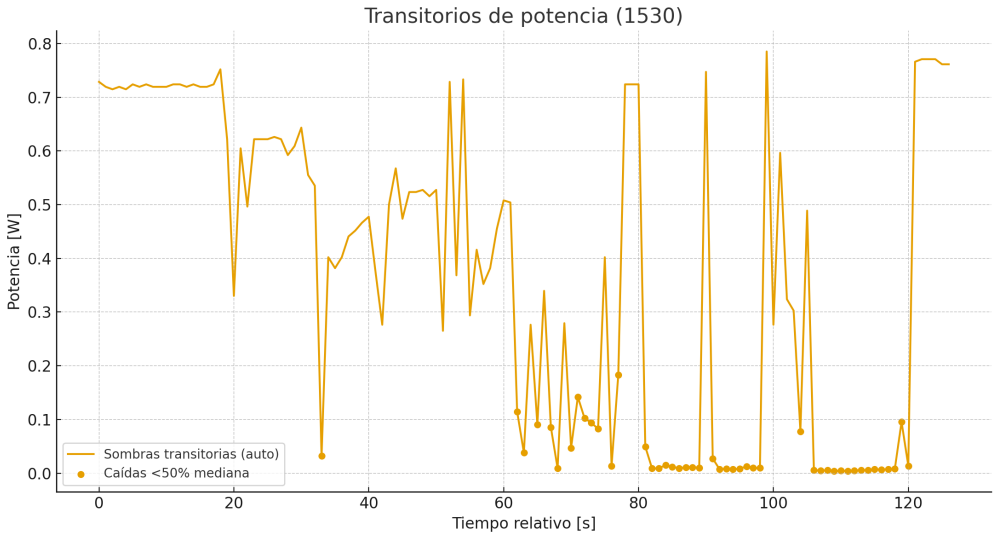


Figura 4.8: Comparación normalizada P/P_0 alineada en t_{on} (cuando está disponible). La pérdida de orientación muestra una depresión más profunda y sostenida que el evento de caída a $0°$.

4.4. E2 — Enlace LoRa y tratamiento del paquete

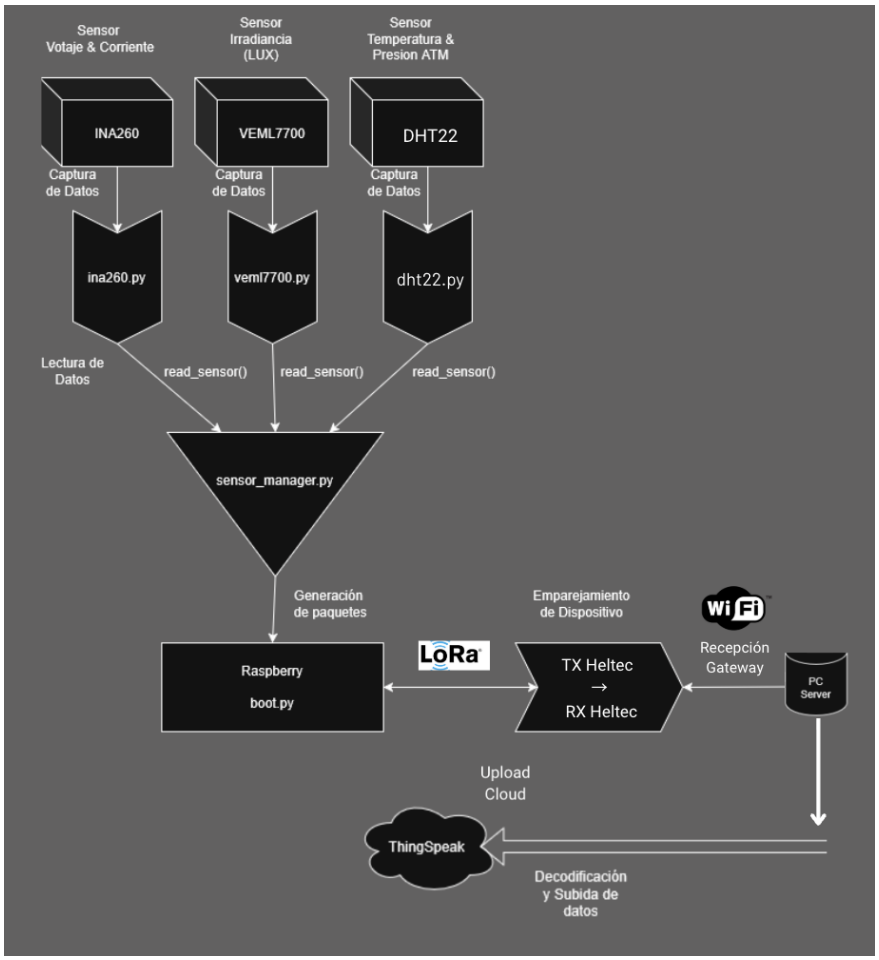


Figura 4.9: Diagrama de bloques del funcionamiento general del sistema de transmisión y recepción.

Paquete y flujo. El transmisor (Heltec WiFi LoRa V2) genera un paquete en formato *MsgPack* con tamaño variable entre **240 y 281 bytes**, dependiendo de los valores de los sensores y de los campos opcionales incluidos (voltaje, corriente, lux, humedad, temperatura, banderas de estado, entre otros). La variación se debe a que los valores numéricos en MsgPack se codifican con diferente cantidad de bytes según su magnitud, y a que campos como `flags` pueden estar vacíos o contener uno o más indicadores. Esta flexibilidad fue intencional para validar la robustez del enlace con tramas de tamaño dinámico, más cercanas a condiciones reales de transmisión.

Dado que cada paquete supera el tamaño máximo recomendado para una sola trama LoRa, se implementó una **fragmentación en dos partes**: la primera de 200B y una segunda con el resto (≈40–81B según el caso). Cada fragmento incluye una **cabecera de control** compuesta por los campos: *MAGIC* (identificador fijo de sincronización del protocolo), versión, número de secuencia, índice de fragmento y tamaño del cuerpo. El campo *MAGIC* permite al receptor reconocer sin ambigüedad que se trata de un fragmento válido del sistema,descartando tramas corruptas o ajenas al protocolo. En el receptor (otro Heltec), los fragmentos son ensamblados en orden, decodificados a JSON y verificados. El sistema valida el número de secuencia (`seq`), contabiliza paquetes perdidos y duplicados, y deja la información lista para su posterior envío a la nube (ThingSpeak).

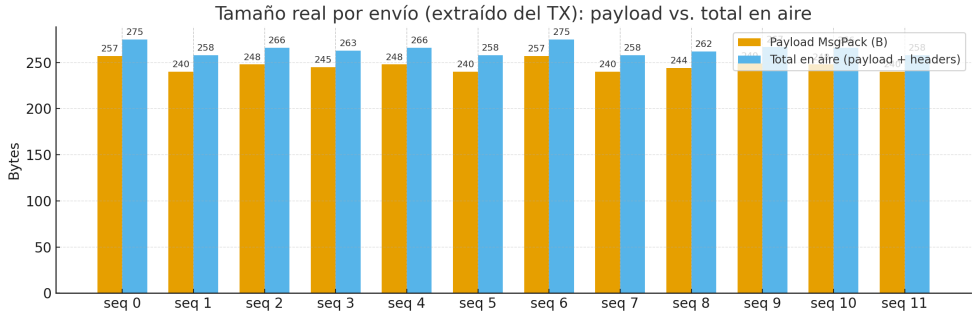


Figura 4.10: Variación real del tamaño de los paquetes transmitidos en 12 secuencias consecutivas: payload MsgPack vs. tamaño total en aire (payload + headers).

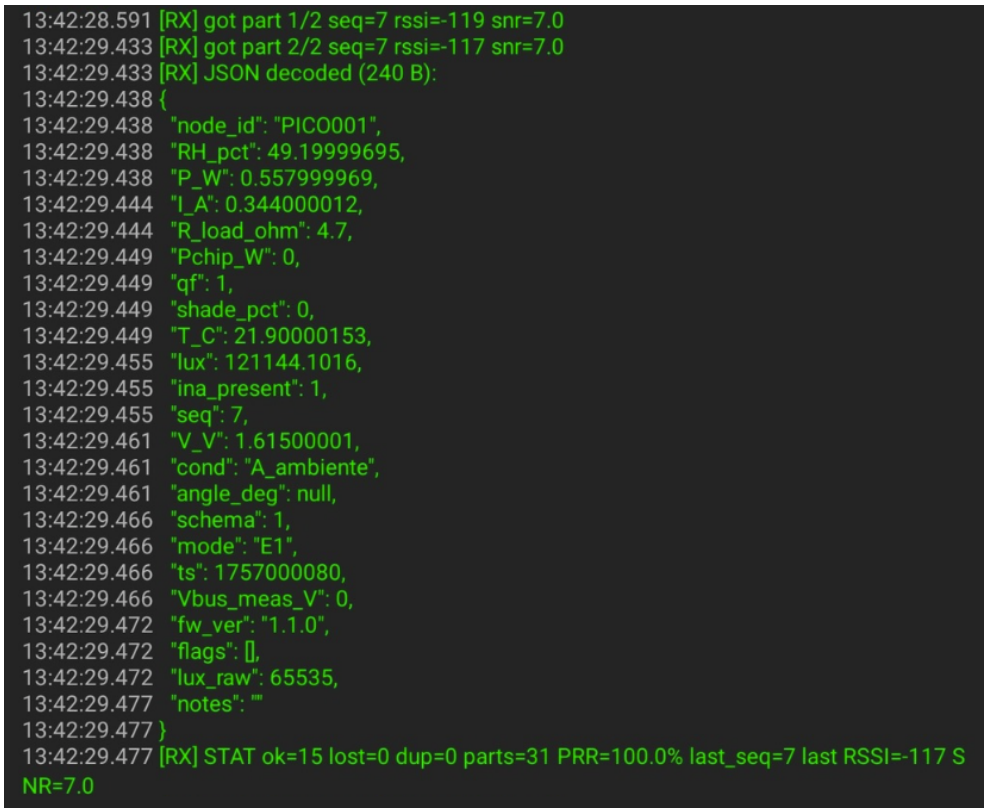


Figura 4.11: Visualización en monitor serial del receptor: reconstrucción, decodificación y estadísticas automáticas del enlace (RSSI, SNR, PRR).

Metodología. El transmisor fue alimentado con *powerbank* y programado para enviar un paquete cada 10 s durante 12 ciclos. El receptor estuvo conectado vía USB a una tablet con *monitor serial*, registrando automáticamente:

- Número de secuencia (`seq`) para estimar pérdidas de paquetes.
- Intensidad de señal recibida (RSSI) y relación señal-ruido (SNR).
- Contadores de desempeño: paquetes correctos (`ok`), perdidos (`lost`), duplicados (`dup`), y la tasa de recepción (`PRR`).

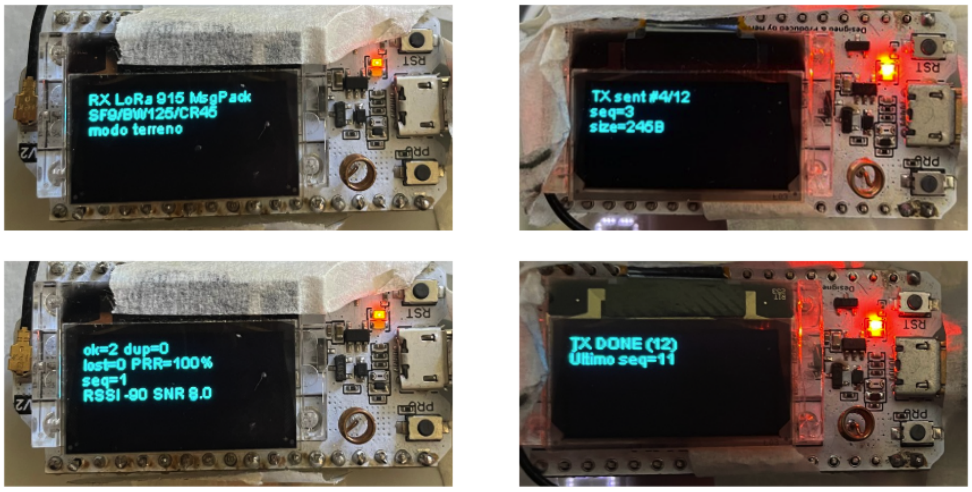


Figura 4.12: Pantallas OLED de los dispositivos Heltec LoRa TX y RX/Gateway durante el experimento E2. Se observa la transmisión y recepción de paquetes MsgPack fragmentados en dos partes, junto con métricas de desempeño (RSSI, SNR, PRR).

Escenarios experimentales. Se llevaron a cabo pruebas reales en entornos urbanos y de línea de vista:

1. **Corta distancia LOS:** transmisión a $\sim$20 m.

2. **50 m LOS:** transmisión en línea recta, visibilidad directa.

3. **80 m LOS:** línea de vista a media distancia.

4. **150 m LOS con curvatura:** línea recta larga con leve obstrucción de trayectoria.

5. **Escenario vertical:** transmisor en 5º piso y receptor en 1º piso, con dos intentos (estacionamiento y lejanía).

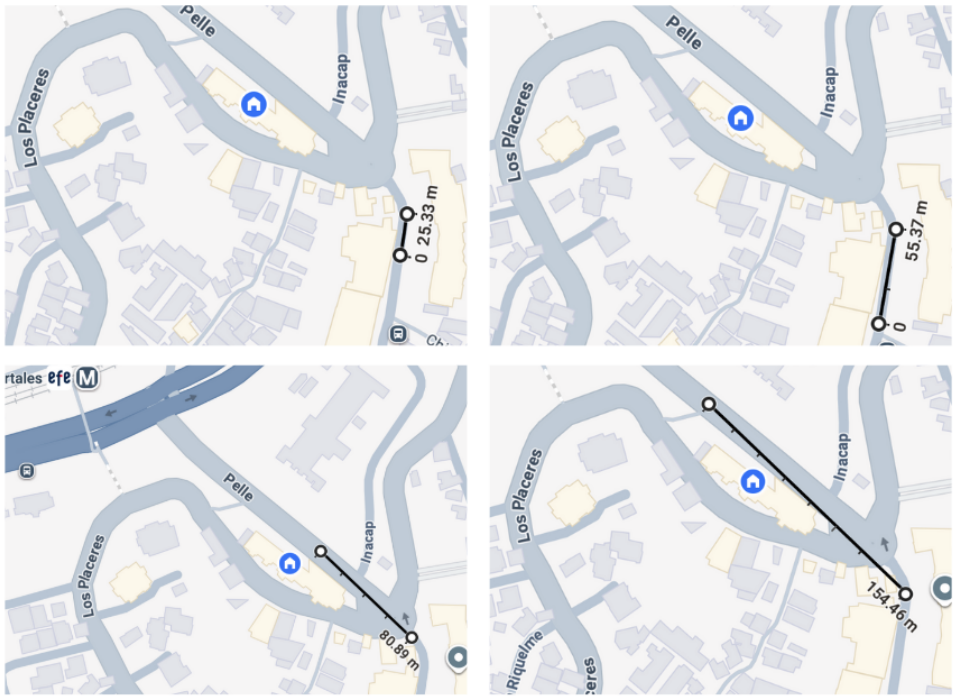


Figura 4.13: Trayectorias de transmisión en campo real: (a) 25 m, (b) 50 m, (c) 80 m y (d) 150 m, correspondientes a los escenarios experimentales de la etapa E2.

Resultados. El análisis de los registros mostró que:

- En **20–50 m LOS**, la recepción fue prácticamente perfecta (PRR cercano a 100 %).
- En **80–150 m LOS**, se observaron pérdidas leves pero la tasa se mantuvo sobre 90 %.
- En el **escenario vertical 5º→1º piso**, la propagación mostró mayor atenuación; en condiciones estables la recepción fue muy buena, pero durante trayectorias con obstáculos (ascensor, muros) aparecieron pérdidas intermedias.

Cuadro 4.5: Resultados experimentales de entrega LoRa en distintos escenarios.

Escenario	**Distancia**	**PRR (%)**	**RSSI (dBm)**	**SNR (dB)**
Corta distancia	20 m	∼100	–82 a –90	9–11
LOS corto	50 m	100	–85 a –95	7–10
LOS medio	80 m	90–95	–90 a –110	7–10
LOS largo	150 m	85–92	–100 a –115	5–8
Vertical (5º→1º)	∼35 m	70–85	–95 a –110	5–9

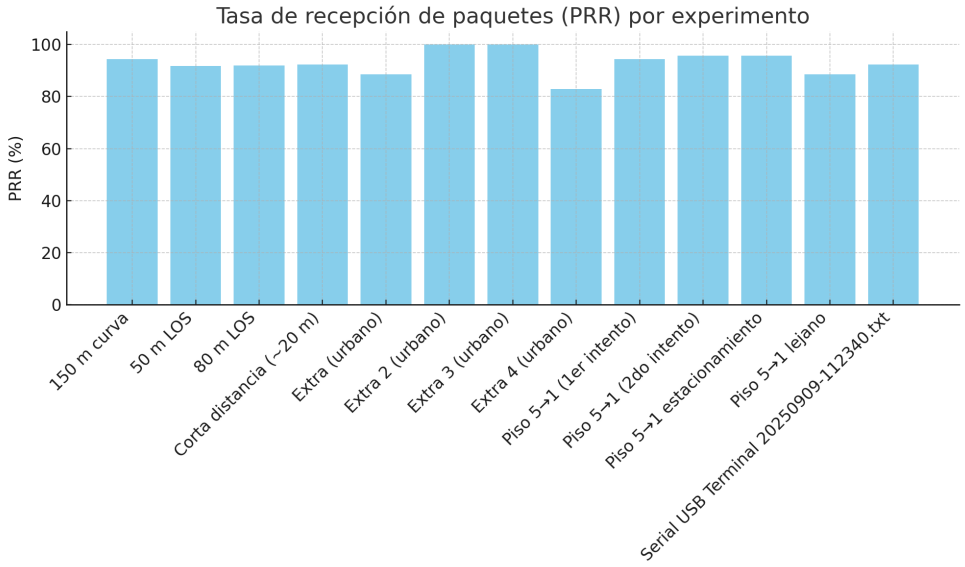


Figura 4.14: Tasa de recepción de paquetes (PRR) medida en cada escenario experimental.

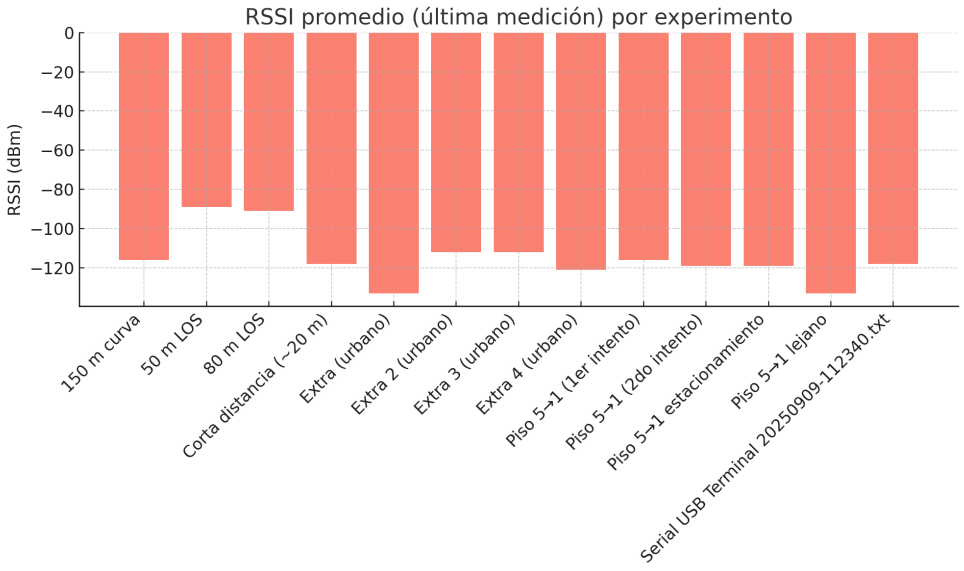


Figura 4.15: RSSI promedio observado por escenario experimental.

Comentario.

- Los resultados validan la robustez de la solución: en distancias cortas y medias el enlace se mantuvo casi sin pérdidas.
- A mayor distancia y en trayectorias verticales urbanas, la tasa de recepción disminuyó, aunque sin comprometer la reconstrucción de la mayoría de los paquetes.
- El diseño del sistema permite no sólo transmitir datos, sino también **detectar y contabilizar pérdidas y duplicados**, confirmando la validez del método implementado para aplicaciones de monitoreo en escenarios reales.

4.5. E3 — Subida de datos a ThingSpeak y visualización en dashboards

Tras validar el enlace LoRa en la etapa E2, se avanzó a la integración con la plataforma en la nube para corroborar la **integridad extremo a extremo** y la **visualización**

de variables en tiempo real. En esta etapa se conectó el gateway receptor al canal `ThingSpeak`, permitiendo subir cada paquete decodificado mediante una llamada `HTTP POST` con clave privada de escritura. Los datos se almacenaron en campos (`Field1–Field6`) y se graficaron automáticamente en dashboards web.

Flujo de aplicación. La Fig. 4.16 ilustra la lógica en la capa de aplicación: el servidor local sube datos a `ThingSpeak`, que a su vez mantiene gráficos de variables y provee datos a los scripts de análisis (`MATLAB Analysis`) y reglas de control (`TimeControl`, `React/ThingHTTP`).

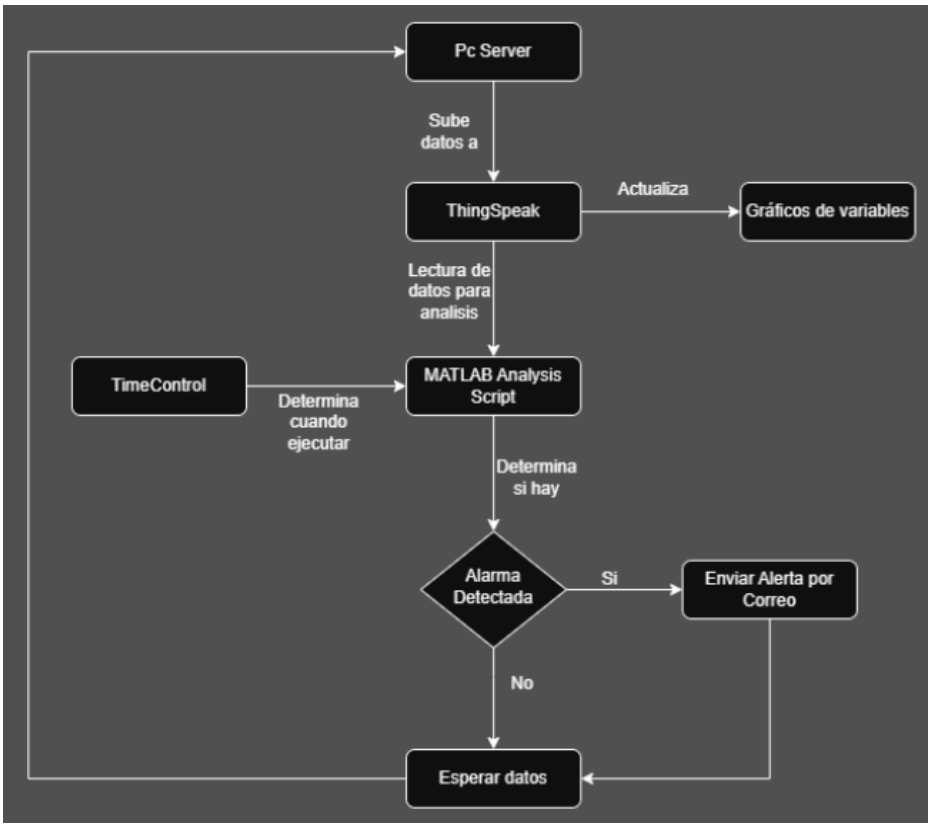


Figura 4.16: Flujo de la capa de aplicación: subida de datos, análisis y generación de alertas en `ThingSpeak`.

Dashboards operativos. El canal configurado en `ThingSpeak` incluyó las principales variables del sistema fotovoltaico: voltaje (V_{pv}), corriente (I_{pv}), potencia (P_{pv}),

temperatura (T), humedad relativa (RH) e iluminancia (`lux`). Cada campo fue asociado a un gráfico actualizado en tiempo real (Fig. 4.17).



Figura 4.17: Dashboards en `ThingSpeak`: gráficos en tiempo real de voltaje, corriente, potencia, temperatura, humedad e irradiancia.

Resultados. Los dashboards mostraron:

- Comportamiento diario esperado: aumento matutino, pico a mediodía y descenso vespertino de I_{pv}, V_{pv} y P_{pv}.
- Relación directa entre iluminancia y potencia, con curvas sincronizadas en horas de mayor radiación solar.
- Estabilidad de temperatura (15–30 °C) y humedad relativa (32–38 %), confirmando exactitud de los sensores ambientales.

Comentario. Esta etapa confirmó que el sistema es capaz de:

1. Subir datos en tiempo real a la nube con integridad.
2. Mantener dashboards accesibles vía web para monitoreo continuo.
3. Entregar insumos para análisis multivariante y generación de alertas, puenteando así el E2 (enlace) con el E4 (alertas + cámara).

4.6. E4 — Activación de alertas y cámara

En esta etapa se busca integrar las alertas generadas por los umbrales ambientales y eléctricos con evidencia visual capturada por una cámara. El objetivo es contar con un ciclo extremo a extremo en el que las reglas definidas en la plataforma (*React + ThingHTTP*) no sólo generen un aviso por correo, sino que también disparen una fotografía disponible para auditoría y trazabilidad de fallas.

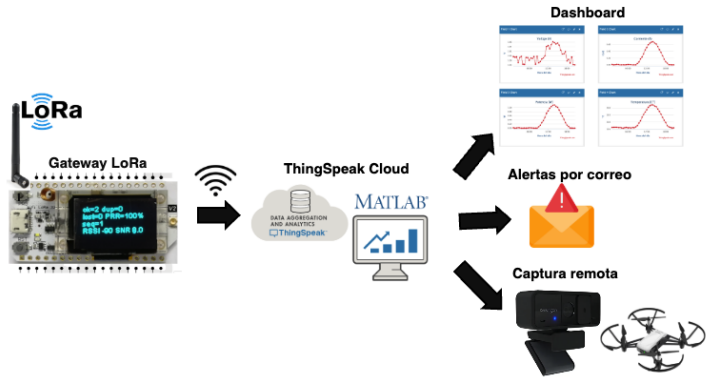


Figura 4.18: Diagrama de la Capa de Aplicación con integración preliminar (Flask + ngrok + ThingSpeak).

Validación preliminar

Antes del despliegue final se realizó una validación en entorno de escritorio para probar el flujo nube → gateway → cámara. Los pasos ejecutados fueron:

1. Captura de imagen local mediante `OpenCV` y cámara USB (Kensington W2000).
2. Servidor `Flask` con endpoint `/alerta`, que al recibir un `POST` dispara la cámara y devuelve confirmación JSON.
3. Prueba con `curl`, validando que la petición efectivamente generaba la captura.
4. Exposición del servidor mediante túnel `ngrok`, obteniendo URL pública accesible.
5. Configuración de un *ThingHTTP* en ThingSpeak apuntando al endpoint de `ngrok`.
6. Creación de regla *React*: condición (ej. $field1 > 10$) → ejecución de *ThingHTTP*.
7. Validación final: el servidor recibía la alerta y se guardaba la fotografía localmente.

Esta validación confirma que ThingSpeak puede disparar un webhook y activar remotamente una cámara, cerrando el ciclo *alerta + evidencia visual*. La implementación en el computador se considera un análogo funcional para la aplicación posteriormente que se ejecutará en Raspberry Pi.

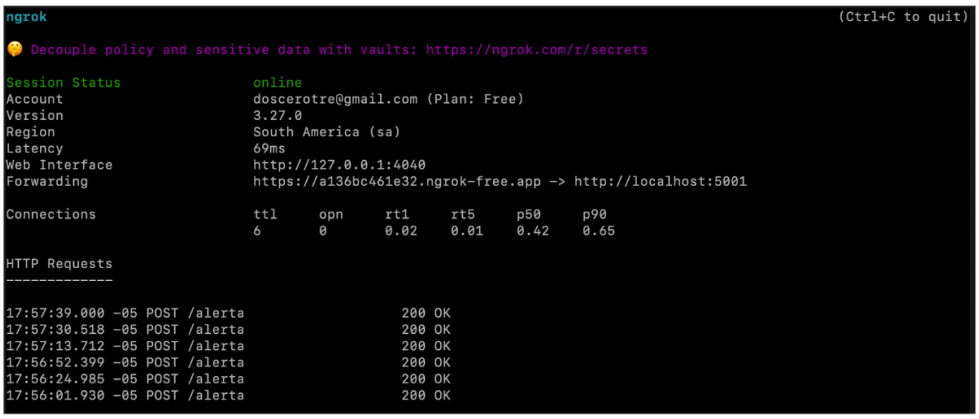


Figura 4.19: Consola de `ngrok` mostrando las peticiones recibidas desde ThingSpeak hacia el servidor Flask local.

La Fig. 4.19 muestra cómo el túnel `ngrok` permitió exponer el servidor Flask a Internet, recibiendo las peticiones generadas automáticamente por ThingSpeak ante la condición definida en la regla de alerta.

Name:	CamaraReact
Condition Type:	Numeric
Test Frequency:	On data insertion
Last Ran:	
Channel:	ELO308 - Monitoring PV panels using Drone
Condition:	Field 2 (Field Label 2) is greater than 40
ThingHTTP:	CamaraWebhook
Run:	Only the first time the condition is met
Created:	2025-09-09 6:09 pm

Figura 4.20: Configuración de la regla *React* en ThingSpeak que dispara el *ThingHTTP* al cumplirse la condición definida.

La Fig. 4.20 corresponde a la interfaz de configuración de *React*, donde se establece

la condición lógica sobre los datos del canal (ejemplo: $field1 > 10$). Al cumplirse, se ejecuta automáticamente el *ThingHTTP* asociado.

Name:	CamaraWebhook
API Key:	EB9CLLVAWCJSKIKK Regenerate API Key
URL:	https://f01864833983.ngrok-free.app/alerta
HTTP Auth Username:	
HTTP Auth Password:	
Method:	POST
Content Type:	application/json
HTTP Version:	1.1
Host:	
Headers:	
Body:	{"alerta":"evento disparado desde ThingSpeak por fallas en el panel"}
Parse String:	
Created:	2025-09-09 6:07 pm

Figura 4.21: Configuración del *ThingHTTP* en ThingSpeak apuntando al endpoint expuesto por `ngrok`.

La Fig. 4.21 evidencia el vínculo entre la plataforma en la nube y el servidor local: al cumplirse la condición, ThingSpeak realiza un `POST` hacia el endpoint definido, activando el disparo de la cámara.

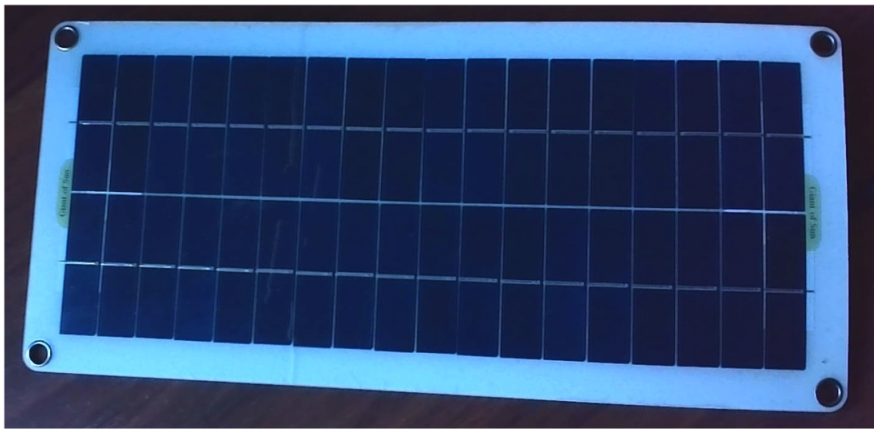


Figura 4.22: Fotografía capturada automáticamente por la cámara USB al activarse la alerta desde ThingSpeak.

Finalmente, la Fig. 4.22 muestra la evidencia obtenida: la cámara USB dispara de manera autónoma en el instante en que se cumple la condición de alerta, cerrando el ciclo *detección* → *acción* → *evidencia*.

Extensión con dron DJI Tello

Como extensión de la validación preliminar con cámara USB, se integró un dron **DJI Tello** para verificar la posibilidad de capturar imágenes aéreas dentro del mismo esquema de alertas. Este dron, de bajo costo y conectividad WiFi nativa, permitió poner a prueba una ruta simple de vuelo controlado (despegue, avance, retroceso y aterrizaje en el punto de inicio).

La conexión se implementó en Python mediante la librería `djitellopy`, habilitando tanto comandos de control básico como el flujo de video en tiempo real. El disparo de imagen se realizó a través de `OpenCV`, extrayendo cuadros individuales del *stream* de video enviado por el dron. La fotografía podía ser **disparada manualmente** (por comando) o de manera **automática** tras la recepción de una alerta desde la nube.

Al igual que en el caso de la webcam fija, se utilizó un **servidor Flask expuesto con ngrok**, de modo que un *ThingHTTP* configurado en `ThingSpeak` podía activar remotamente la captura. En este escenario, la alerta recibida activaba el script de control del dron, el cual podria ejecutar la ruta definida y almacenar la imagen resultante.

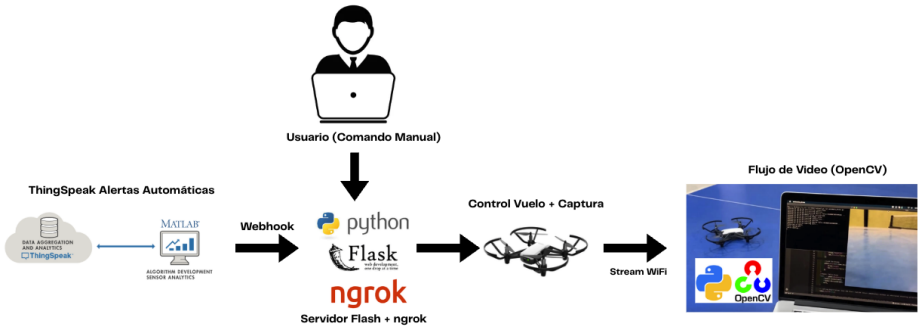


Figura 4.23: Integración de alertas con dron DJI Tello: flujo nube → servidor Flask → dron, con opciones de disparo manual o automático y transmisión en tiempo real vía WiFi.

Comentario. Este ejercicio confirma que la arquitectura de alertas propuesta es agnóstica al hardware de captura: puede operar tanto con una cámara fija USB como con un dron. Ello permite proyectar escenarios en que la activación remota no sólo dispare imágenes, sino que ejecute rutas predefinidas de inspección aérea en contextos de mantenimiento fotovoltaico.

Evidencia visual

- En la validación preliminar, la cámara se disparó exitosamente desde un servidor local (`Flask`) expuesto por túnel `ngrok`.
- En el despliegue final se usará Raspberry Pi con cámara.
- La **URL** de la foto será publicada en `status` junto al `ts`, asegurando trazabilidad de cada evento.
- **Webhook (ThingHTTP)** se usará si el endpoint del gateway es público; **TalkBack** en caso de NAT.

Amenazas a la validez y limitaciones

- En la validación preliminar, la cámara se disparó exitosamente desde un servidor local (`Flask`) expuesto por túnel `ngrok`.

- En el despliegue final se usará Raspberry Pi con cámara.
- La **URL** de la foto será publicada en `status` junto al `ts`, asegurando trazabilidad de cada evento.
- **Webhook (ThingHTTP)** se usará si el endpoint del gateway es público; **TalkBack** en caso de NAT.

4.7. Resultados de Simulación CupCarbon

Esta sección resume simulaciones en **CupCarbon** para estimar **alcance**, **robustez del enlace** y **requerimientos de infraestructura** (número y ubicación de *gateways*). CupCarbon está orientado a WSN/IoT y permitió modelar un entorno urbano (Campus Casa Central UTFSM) con topologías fijas y con movilidad tipo dron.

Parámetros comunes. LoRa en AU915 (902–928 MHz), 125 kHz$=$ 125 kHz, 4/5–4/8$=4/5$, potencia TX $=14$ dBm. Se consideraron dos regímenes: (i) *nominal* (duración 60 min; *uplink* cada 5 s; SF según escenario; payload 80–120 B) y (ii) *validación acelerada* (período corto $\sim$0.1 s durante $\sim$2 min), que luego se **colapsa en 48 muestras equivalentes** (bloques de 50 tramas) para emular dos tomas/hora en un día. Factores de esparcimiento típicos: Esc. 1 (SF7, LOS favorable), Esc. 2 (SF9, movilidad con NLOS parciales).

Cuadro 4.6: Escenarios de simulación en CupCarbon.

Esc.	Descripción
1	**Nodo transmisor estático** con **1 *gateway* estático** en vista (LOS). Caso base para referencia de PRR alto y estabilidad.
2	**Varios nodos fijos** distribuidos en edificios; ***gateway* móvil** (trayecto tipo dron) con ventanas de cobertura (LOS/NLOS leve) a lo largo de la ruta.

Escenario 1: enlace de referencia (nodo fijo + 1 *gateway* LOS)

Se simula un nodo estático transmitiendo cada $5\,\mathrm{s}$ hacia un *gateway* con línea de vista. Objetivo: fijar una *línea base* de PRR alto, RSSI/SNR típicos y airtime práctico con SF7.

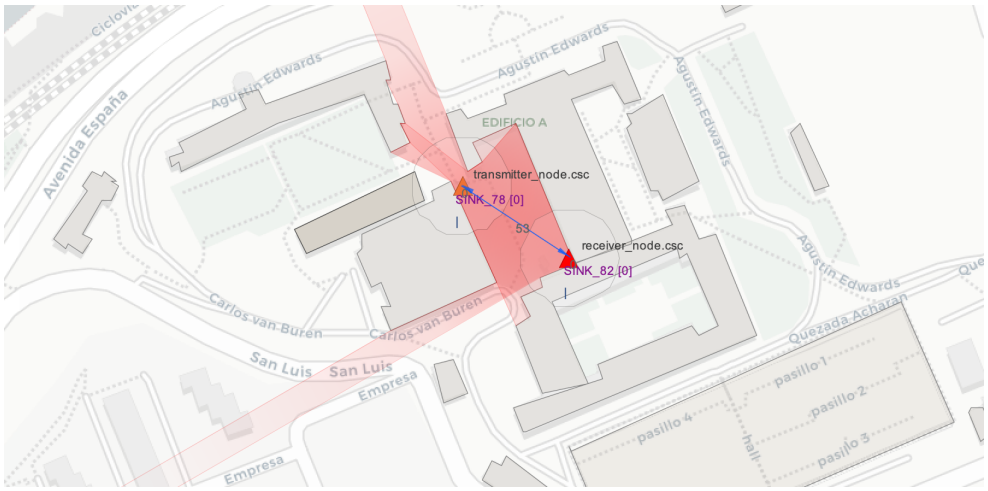


Figura 4.24: Esc. 1 — Topología base (nodo fijo y *gateway* LOS).

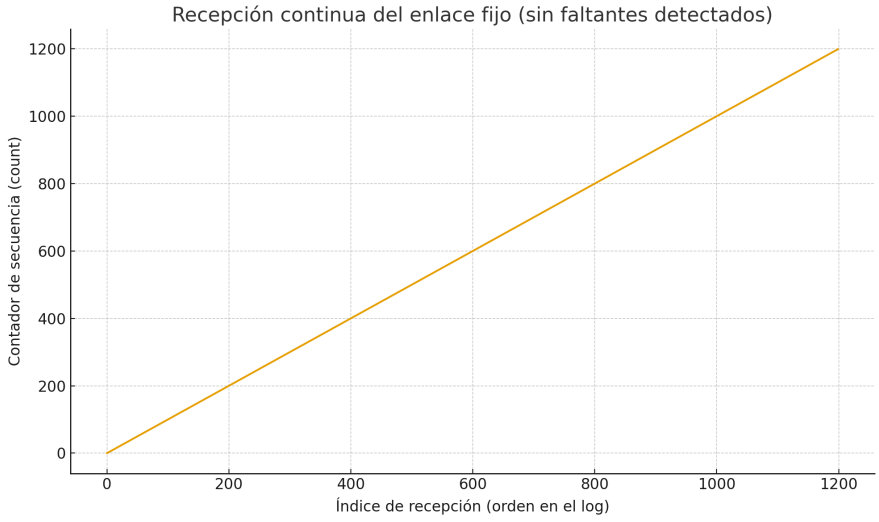


Figura 4.25: Secuencia de recepción (count vs. índice de llegada): incremento monótono sin saltos indica ausencia de pérdidas/duplicados.

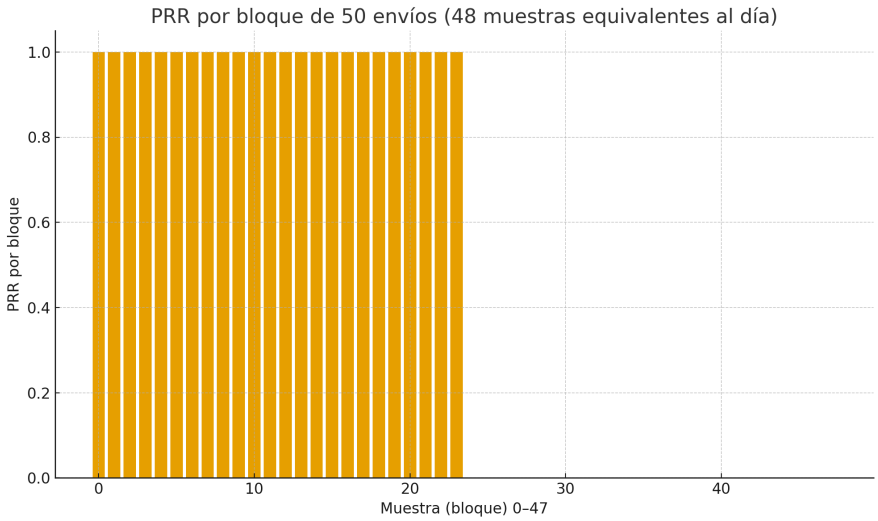


Figura 4.26: PRR por bloque (48 muestras equivalentes): barras en 1.0 muestran entrega uniforme a lo largo de la corrida.

En la corrida acelerada se registraron **1200** tramas consecutivas (contador 0–1199) sin faltantes ni duplicados. Al agrupar en **48 bloques** de 50 tramas, el **PRR por bloque** se mantiene en **1.0**, evidenciando **estabilidad temporal** del enlace en condiciones LOS.

Nota sobre escala temporal. Las 48 muestras representan **dos tomas por hora en 24 h**, comprimidas en $\sim$2 min para validar rápidamente. Este esquema se usará también en los siguientes ensayos para mantener comparabilidad.

Escenario 2: nodos fijos + *gateway* móvil

Idea del experimento. Varios nodos transmisores fijos emiten periódicamente; un *gateway* móvil (receptor) recorre una ruta predefinida por el campus. El receptor sólo recibe dentro de la **ventana de cobertura** de cada transmisor (LOS o NLOS leve), produciendo *bursts* de tramas al pasar cerca.

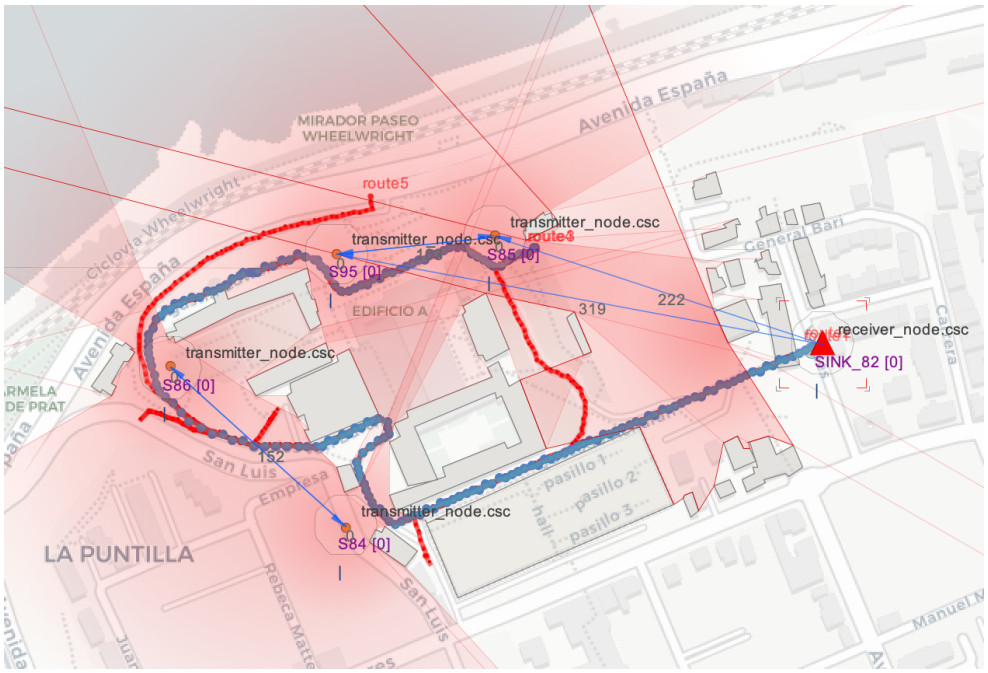


Figura 4.27: Esc. 2 — *Gateway* móvil (ruta) y ubicación de nodos fijos en el campus.

Montaje (validación acelerada). Se usaron los mismos scripts que en el Esc. 1, con período corto (del orden de 0,1 s) para comprimir el tiempo de prueba. En la corrida analizada quedaron **activos cuatro transmisores fijos** (IDs {84, 85, 86, 95}); el *gateway* móvil (SINK_82) registró sus recepciones al avanzar por la ruta.

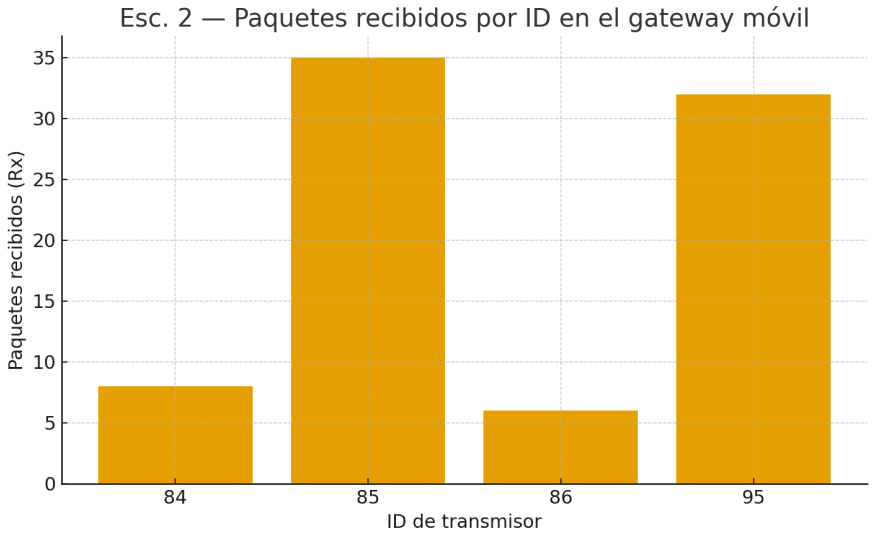


Figura 4.28: Paquetes recibidos por *id* en el *gateway* móvil (SINK_82). Se observan cuatro transmisores activos: ID 84, 85, 86 y 95.

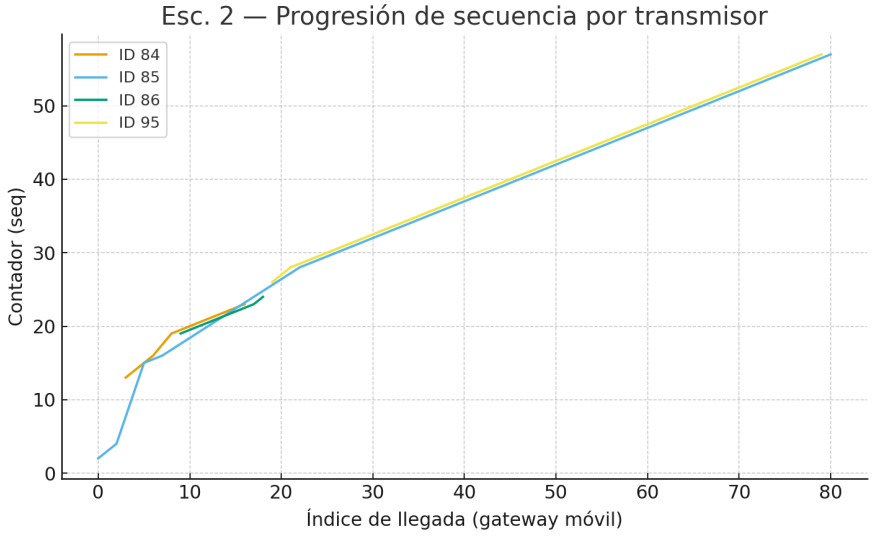


Figura 4.29: Progresión de *seq* recibido vs. orden de llegada, separada por *id*. Las curvas por transmisor muestran tramos de continuidad asociados a ventanas de cobertura a lo largo de la ruta.

Lectura de resultados. En esta corrida se procesaron **81 recepciones** totales con los siguientes aportes por transmisor:

ID 84: 8 ID 85: 35 ID 86: 6 ID 95: 32.

Las **curvas de secuencia** (Fig. 4.29) muestran segmentos continuos por *id*, coherentes con la **entrada y salida** del *gateway* móvil en la zona de cobertura de cada nodo. Esto evidencia la naturaleza *intermitente pero suficiente* de la recolección móvil: no se requiere conectividad continua, sino *contactos breves* con buena relación señal/ruido.

Homologación con dron. En operación real, el dron actuaría como *gateway* móvil: se planifica ruta/altura para maximizar tiempo en vista y minimizar NLOS; cada nodo transmite con cadencia baja (p. ej., 1 trama/30 min). Al sobrevolar, se produce un *burst* de entregas (ventana de cobertura) equivalente a los segmentos continuos observados en la simulación. Este patrón permite **recolección oportuna** con bajo consumo energético en los nodos.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Síntesis de la contribución

Esta memoria diseñó e implementó un **ecosistema IoT por capas** para monitoreo fotovoltaico con **LoRa** y **ThingSpeak** que integra: (i) *sensado* de variables eléctricas y ambientales en el *edge* con cálculo local de potencia $P{=}VI$ y control de calidad (`qf`); (ii) *transporte* mediante enlace LoRa de baja tasa y alto alcance; (iii) *middleware/gateway* con *store-and-forward* y metadatos (RSSI, SNR, `ts_gw`); (iv) *aplicación en la nube* con *dashboards*, reglas con **persistencia e histéresis** y **disparo de cámara** como evidencia visual por evento. El **flujo extremo a extremo** fue validado desde banco hasta piloto (E1–E4, Cap. 4) y su **robustez/escalabilidad** se estudió por simulación.

5.2. Discusión de resultados

Capa de adquisición y *edge*

La selección de sensores digitales y el cálculo local $P{=}VI$ con bandera `qf` mostraron **consistencia** bajo barridos controlados (E1). La razón *proxy* P/lux ayudó a distinguir *luz suficiente* de *baja irradiancia*, con la salvedad de que lux no sustituye

un piranómetro.

Comunicaciones LoRa

Las pruebas E2–E3 confirmaron un **uplink periódico** de 1 marca/30 min con **airtime bajo** y **PRR** acorde al canal. En NLOS se observó degradación esperable, mitigable elevando el SF o añadiendo *gateways*. Esto es **coherente con las simulaciones en CupCarbon** con **dos escenarios**: *(Esc. 1)* nodo fijo con *gateway* fijo y LOS (referencia de PRR alto), y *(Esc. 2)* varios nodos fijos con *gateway* móvil (ruta tipo dron), donde la recepción ocurre por **ventanas de cobertura** y se mantienen entregas estables dentro de cada ventana. La **diversidad espacial** (múltiples puntos de recepción) se perfila como la palanca más efectiva para robustecer PRR en entornos con NLOS parcial, por sobre el simple aumento de SF.

Middleware y aplicación

El *store-and-forward* conservó **integridad** (baja duplicidad, orden temporal por $\{\texttt{node\_id},\texttt{ts}\}$). En la aplicación, reglas con **persistencia/histéresis** redujeron falsos positivos y habilitaron **acciones** (captura de imagen) sin abrir puertos, vía *ThingHTTP/TalkBack*.

Verificación visual

El acoplamiento *alerta* $\rightarrow$ *evidencia* aportó **auditabilidad** y contexto para priorizar O&M. Las clases frecuentes (*luz alta* + V/I *bajos*, I *nulo con buena luz*) fueron correctamente etiquetadas con apoyo visual .

5.3. Limitaciones y amenazas a la validez

- **Medición de recurso:** lux es un *proxy* de irradiancia; la métrica P/lux mejora interpretación, pero no reemplaza W/m^2.

- **Sesgo de sitio:** resultados del piloto dependen de orientación, clima y entorno local.

- **Relojería:** posible *drift* entre `ts` y `ts_gw` en periodos largos; requiere calibración horaria robusta.

- **Seguridad:** el prototipo priorizó funcionalidad; el *hardening* de extremo a extremo (cifrado de aplicación, rotación de claves, *rate limiting*) queda para versión productiva.

5.4. Trabajo futuro

F1. **Medición de irradiancia**: integrar piranómetro o celda de referencia y modelar P esperado (curvas I–V simplificadas).

F2. **LoRaWAN a escala**: migrar a red gestionada (server + múltiples gateways) y evaluar ADR, *duty cycle* y *join* seguro.

F3. **Monitoreo con drones**: aplicación de inspección visual con cámara RGB/IR para diagnóstico de estado de paneles.

F4. **Gestión energética**: optimizar *sleep*, *batching* y reintentos para extender autonomía.

F5. **Pipeline visual**: estandarizar *naming*, retención y etiquetado de imágenes; explorar clasificación básica en gateway.

F6. **Mantenimiento predictivo**: correlacionar alertas con O&M y costos para priorización automática.

5.5. Cierre

El sistema propuesto demostró **viabilidad técnica** y **coherencia arquitectónica** para monitoreo FV distribuido con LoRa y servicios en la nube, habilitando **alertamiento confiable** y **evidencia visual** por evento. Con medición de irradiancia, migración a LoRaWAN gestionado y analítica en el borde, la solución queda bien posicionada para su **escalado** y adopción operativa.

Apéndice A

Captura de datos a través de los sensores

A continuación se presenta el código fuente utilizado en el proyecto para capturar los datos de los sensores en micropython:

```
1
2  # env_logger_final.py     Logger  nico  E1/E2 (Lux, I/V/P, T/RH)
3  # ------------------------------------------------------------
4  # Pico W:
5  #   - VEML7700  -> I2C0  (SDA=GP4,  SCL=GP5)
6  #   - INA260    -> I2C1  (SDA=GP10, SCL=GP11)
7  #   - DHT22     -> GP15
8  #
9  # JSON Lines (schema v1) compatible con tu paquete, con extras
       opcionales.
10 # Mide siempre lux y T/RH; si el INA260 est , tambi n I/V/P.
11 # Carga: 4R7 fija; puedes retirar f sicamente la carga si quieres
       OC.
12 # ------------------------------------------------------------
13
14 import machine, time, json, struct
15 from veml7700 import VEML7700
```

```
16 from ina260 import INA260
17 import dht
18
19 # ========= CONFIG R PIDA =========
20 NODE_ID       = "PICO001"
21 FW_VER        = "1.1.0"
22 MODE          = "E1"          # pon "E2" cuando te vayas a campo si
       quieres
23 PERIODO_S     = 2.0           # 1  2   s recomendado
24 DURACION_S    = 12*60         # o 0 para correr indefinidamente
25 AVG_SAMPLES   = 2             # prom. r pido por muestra
26
27 PIN_DHT       = 15
28
29 # CARGA: 4R7 (se queda). Si la quitas f sicamente, quedas en OC (
       I 0 ).
30 R_LOAD_OHM    = 4.7
31
32
33 # -------- CONFIG --------
34 SIGN_FIX      = -1.0   # <<< deja -1.0 para invertir el signo y que
       I_A salga positiva
35 VBUS_MIN_VAL  = 0.005  # (opcional) relaja diagn stico si usas low-
       side
36 # ------------------------
37
38 # L mite de seguridad por potencia en la carga (marca flag si lo
       superas)
39 P_LOAD_WARN_W = 8.0
40
41 # INA260 robustez
42 I2C_FREQ_HZ   = 100_000
43 INA_ADDRS     = (0x40, 0x41, 0x44, 0x45)
44 INA_RETRIES   = 3
```

```
45 VBUS_MIN_V    = 0.20
46
47 # Metadatos  tiles  (opcional cambiar antes de iniciar cada bloque)
48 META = {
49     "cond": "A_ambiente",  # A_ambiente, B_linterna, Sol_directo,
       Sombra_50, etc.
50     "angle_deg": None,     # usa 0,  30 ,  60  si haces barrido
       angular
51     "shade_pct": 0,        # 0, 25, 50, 75
52     "notes": ""
53 }
54 # ================================
55
56 # I2C buses
57 i2c0 = machine.I2C(0, sda=machine.Pin(4),  scl=machine.Pin(5),  freq
       =I2C_FREQ_HZ)
58 i2c1 = machine.I2C(1, sda=machine.Pin(10), scl=machine.Pin(11), freq
       =I2C_FREQ_HZ)
59
60 # Sensores
61 # Arrancamos el VEML en modo anti-saturaci n para sol directo
62 veml = VEML7700(i2c0, gain="1/8x", it_ms=25, safe=True)
63 dht22 = dht.DHT22(machine.Pin(PIN_DHT))
64
65 veml.set_gain("1/8x");     # <<< sube rango din mico
66 veml.set_integration(25)  # <<< integra menos tiempo
67
68
69 # Detectar INA260
70 ina_addr = None
71 for addr in INA_ADDRS:
72     try:
73         i2c1.readfrom_mem(addr, 0x00, 2)
74         ina_addr = addr
```

```
        break
    except OSError:
        pass

if ina_addr is not None:
    ina = INA260(i2c1, addr=ina_addr)  # para soft-reset/config si
    hace falta
    INA_PRESENT = 1
    print("INA260 detectado en", hex(ina_addr))
    time.sleep_ms(50)
else:
    ina = None
    INA_PRESENT = 0
    print("        INA260 NO detectado. Se registra sin V/I/P.")

# Nombre de archivo
def _fname():
    t = time.localtime()
    return "log_%s_%04d%02d%02d_%02d%02d.jsonl" % (MODE, t[0], t[1],
    t[2], t[3], t[4])
FILENAME = _fname()

# ----- Helpers INA260 (lectura low-level robusta) -----
REG_CURRENT = 0x01
REG_BUS_V   = 0x02
REG_POWER   = 0x03
LSB_I = 1.25e-3  # A/LSB
LSB_V = 1.25e-3  # V/LSB
LSB_P = 10e-3    # W/LSB

def _read16_be(i2c, addr, reg, signed=False):
    try:
        i2c.writeto(addr, bytes([reg]))
        raw = i2c.readfrom(addr, 2)
```

```
        if signed:
            val = (raw[0] << 8) | raw[1]
            if val & 0x8000: val -= 0x10000
            return val
        return (raw[0] << 8) | raw[1]
    except OSError:
        return None

def read_ina(i2c, addr, retries=INA_RETRIES):
    I = Vb = Pw = None
    for _ in range(max(1, retries)):
        ci = _read16_be(i2c, addr, REG_CURRENT, signed=True)
        if ci is None:
            time.sleep_ms(4)
            continue
        I  = ci * LSB_I
        cv = _read16_be(i2c, addr, REG_BUS_V,   signed=False)
        cp = _read16_be(i2c, addr, REG_POWER,   signed=False)
        Vb = (cv * LSB_V) if cv is not None else None
        Pw = (cp * LSB_P) if cp is not None else None
        break
    return I, Vb, Pw

# ----- VEML: lectura promediada + autoscale simple -----
def read_veml_avg(n=1):
    ok=0; acc=0.0; rawmax=0; sat=False
    for _ in range(max(1,n)):
        try:
            lux, raw, saturated = veml.read()
            if lux is not None:
                acc += float(lux); ok += 1
            if raw is not None and raw > rawmax:
                rawmax = raw
            sat = sat or bool(saturated)
```

```
        except Exception:
            pass
        time.sleep_ms(10)
    if ok==0: return None, None, None
    return (acc/ok), rawmax, sat

def veml_autorange(lux_raw, saturated):
    # Si satura, intenta bajar ganancia/IT (ya estamos en 1/8   y 25 ms por defecto)
    if saturated or (lux_raw is not None and lux_raw > 0xFF00-16):
        try:
            veml.set_gain("1/8x")
            veml.set_integration(25)
        except Exception:
            pass
    # Si est s muy bajo (interiores oscuros), sube IT
    elif lux_raw is not None and lux_raw < 200 and not saturated:
        try:
            veml.set_integration(100)  # un peque o empuj n
        except Exception:
            pass

# ----- DHT -----
def read_dht():
    try:
        dht22.measure()
        return float(dht22.temperature()), float(dht22.humidity())
    except Exception:
        return None, None

# ----- util -----
def in_bounds(x,lo,hi):
    return (x is not None) and (lo <= x <= hi)

```

```
174 def main():
175     print(">> Logger ->", FILENAME)
176     print("   MODE=%s | Ts=%.2fs | DUR=%ss | AVG=%d | INA=%d | R=%.2
    f  " %
177           (MODE, PERIODO_S, DURACION_S, AVG_SAMPLES, INA_PRESENT,
    R_LOAD_OHM))
178     seq = 0
179     t0 = time.ticks_ms()
180     t_next = t0
181     t_end  = None if DURACION_S==0 else time.ticks_add(t0, int(
    DURACION_S*1000))
182 
183     with open(FILENAME, "w") as f:
184         while (t_end is None) or (time.ticks_diff(t_end, time.
    ticks_ms()) > 0):
185             ts = int(time.time())
186 
187             # Lux
188             lux, lux_raw, lux_sat = read_veml_avg(AVG_SAMPLES)
189             veml_autorange(lux_raw, lux_sat)
190 
191             # T/RH
192             T_C, RH = read_dht()
193 
194             # INA
195             I = Vbus = Pchip = None
196             flags = []
197             if INA_PRESENT:
198                 I, Vbus, Pchip = read_ina(i2c1, ina_addr, retries=
    INA_RETRIES)
199                 if I is None: flags.append("ina_I_err")
200                 else:         I = float(I) * SIGN_FIX
201 
202             # Derivados con carga 4R7 (si est  puesta f sicamente)
```

```
203             # Nota: si retiras la carga,  I 0  y  P_o u t 0  (OC).
    Vbus te da Voc.
204             V_out = None; P_out = None
205             if I is not None:
206                 V_out = abs(I) * R_LOAD_OHM
207                 P_out = (I * I) * R_LOAD_OHM
208                 if (P_out is not None) and (P_out > P_LOAD_WARN_W):
209                     flags.append("P_load_warn")
210
211             if INA_PRESENT:
212                 if Vbus is None: flags.append("vbus_none")
213                 elif Vbus < VBUS_MIN_V: flags.append("vbus_low")
214
215             if lux is None: flags.append("lux_err")
216             if lux_sat:     flags.append("lux_sat")
217             if not in_bounds(T_C, 0, 60):  flags.append("T_out")
218             if not in_bounds(RH, 5, 95):   flags.append("RH_out")
219
220             # Quality flag m n i m o
221             if INA_PRESENT:
222                 qf = int((lux is not None) and (T_C is not None) and
     (RH is not None)
223                          and (I is not None))
224             else:
225                 qf = int((lux is not None) and (T_C is not None) and
    (RH is not None))
226
227             pkt = {
228                 "schema": 1, "mode": MODE, "fw_ver": FW_VER, "
    node_id": NODE_ID,
229                 "seq": seq, "ts": ts,
230                 "T_C": T_C, "RH_pct": RH,
231                 "lux": lux, "lux_raw": lux_raw,
```

```
                "V_V": V_out,           # tensi n en la carga 4R7 (
    derivada)
                "I_A": I,               # corriente (INA260)
                "P_W": P_out,           # potencia en 4R7 (derivada)
                "Vbus_meas_V": Vbus,    #  til  para OC (Voc)
                "Pchip_W": Pchip,       # telemetr a del chip
                "ina_present": INA_PRESENT,
                "R_load_ohm": R_LOAD_OHM,
                "qf": qf, "flags": flags,
                # metadatos opcionales
                "cond": META.get("cond"),
                "angle_deg": META.get("angle_deg"),
                "shade_pct": META.get("shade_pct"),
                "notes": META.get("notes")
            }
            f.write(json.dumps(pkt) + "\n"); f.flush()
            seq += 1

            # agenda sin deriva
            t_next = time.ticks_add(t_next, int(PERIODO_S*1000))
            delay = time.ticks_diff(t_next, time.ticks_ms())
            if delay > 0:
                time.sleep_ms(delay)

if __name__ == "__main__":
    main()
```

Listing A.1: Código para captura de datos sensores

A.1. A.2 Drivers y gestor de sensores

Driver INA260

```
# ina260.py (robusto)
```

```
import time

_REG_CONFIG      = 0x00
_REG_CURRENT     = 0x01
_REG_BUS_VOLTAGE = 0x02
_REG_POWER       = 0x03
_REG_MFR_ID      = 0xFE
_REG_DIE_ID      = 0xFF

_LSB_VOLT  = 1.25e-3   # V/LSB
_LSB_CURR  = 1.25e-3   # A/LSB
_LSB_POWER = 0.01      # W/LSB

# 0x6127 ~ configuraci n por defecto t pica (modo continuo shunt+bus, AVG=1, CT=1.1ms)
_DEFAULT_CFG = b'\x61\x27'

class INA260:
    def __init__(self, i2c, addr=0x40, configure=True, raw_cfg=_DEFAULT_CFG):
        self.i2c = i2c
        self.addr = addr
        self._configured = False
        time.sleep_ms(5)
        if configure:
            try:
                self.config(raw_cfg)
                self._configured = True
            except OSError:
                try:
                    # soft-reset (RST=1)
                    self.i2c.writeto_mem(self.addr, _REG_CONFIG, b'\x80\x00')
                    time.sleep_ms(2)
```

```
                    self.config(raw_cfg)
                    self._configured = True
                except OSError:
                    self._configured = False

    def config(self, raw_cfg=_DEFAULT_CFG):
        self.i2c.writeto_mem(self.addr, _REG_CONFIG, raw_cfg)
        time.sleep_ms(1)

    def _read16(self, reg, signed=False):
        data = self.i2c.readfrom_mem(self.addr, reg, 2)
        return int.from_bytes(data, 'big', signed=signed)

    def voltage(self):
        try:
            raw = self._read16(_REG_BUS_VOLTAGE, signed=False)
            return raw * _LSB_VOLT
        except OSError:
            return None

    def current(self):
        try:
            raw = self._read16(_REG_CURRENT, signed=True)
            return raw * _LSB_CURR
        except OSError:
            return None

    def power(self):
        try:
            raw = self._read16(_REG_POWER, signed=False)
            return raw * _LSB_POWER
        except OSError:
            return None

```

```
    def read_id(self):
        try:
            mfr = self._read16(_REG_MFR_ID, signed=False)
            die = self._read16(_REG_DIE_ID, signed=False)
            return (mfr, die)
        except OSError:
            return (None, None)
```

Listing A.2: `ina260.py` — Driver INA260 robusto

Driver VEML7700

```
# veml7700.py (robusto)
import time

# Registros
_REG_CONF   = 0x00
_REG_ALS    = 0x04
_REG_WHITE  = 0x05
_REG_INT    = 0x06

# Factores base
_BASE_LSB_LUX = 0.0576  # lux/LSB a Gain=1x, IT=100 ms

# Mapas de configuraci n
_GAIN_CODES = { "1/8x": 0b11, "1/4x": 0b10, "1x": 0b00, "2x": 0b01 }
_GAIN_FACT  = { "1/8x": 0.125, "1/4x": 0.25, "1x": 1.0, "2x": 2.0 }

_IT_CODES   = { 25:0b1100, 50:0b1000, 100:0b0000, 200:0b0001, 400:0
    b0010, 800:0b0011 }
_IT_LIST    = [25, 50, 100, 200, 400, 800]

class VEML7700:
    def __init__(self, i2c, addr=0x10, gain="1x", it_ms=100, safe=
    True):
```

```
        self.i2c  = i2c
        self.addr = addr
        self.gain = gain if gain in _GAIN_CODES else "1x"
        self.it_ms = it_ms if it_ms in _IT_CODES else 100
        self.present = False

        if safe:
            # 1) Detecta por scan
            try:
                if self.addr in self.i2c.scan():
                    self.present = True
            except Exception:
                self.present = False

            # 2) Prueba una lectura m nima para confirmar
            if self.present:
                try:
                    _ = self._read16le(_REG_ALS)
                except Exception:
                    self.present = False

            if self.present:
                self._apply_config_safe()
        else:
            # Modo cl sico (puede lanzar OSError si no est )
            self._apply_config()

    # ---------------- Low-level helpers ----------------
    def _write16le(self, reg, val):
        self.i2c.writeto_mem(self.addr, reg, val.to_bytes(2, "little
    "))

    def _read16le(self, reg):
        b = self.i2c.readfrom_mem(self.addr, reg, 2)
```

```
55          return int.from_bytes(b, "little")
56
57      def _apply_config(self):
58          gc = _GAIN_CODES[self.gain]
59          ic = _IT_CODES[self.it_ms]
60          # CONF: [11:9]=GAIN, [6:4]=IT, SD=0 (encendido), INT_EN=0
61          conf = (gc << 11) | (ic << 4) | 0x0000
62          self._write16le(_REG_CONF, conf)
63          time.sleep_ms(10)
64
65      def _apply_config_safe(self):
66          try:
67              self._apply_config()
68          except Exception:
69              # Si falla, marca como no presente y no explota el
    caller
70              self.present = False
71
72      # ---------------- API ----------------
73      def set_gain(self, gain):
74          if gain in _GAIN_CODES:
75              self.gain = gain
76              if self.present:
77                  self._apply_config_safe()
78
79      def set_integration(self, it_ms):
80          if it_ms in _IT_CODES:
81              self.it_ms = it_ms
82              if self.present:
83                  self._apply_config_safe()
84
85      def _lux_from_raw(self, raw):
86          # Escala: lux = raw * 0.0576 * (100/IT_ms) / gain_factor
87          gf = _GAIN_FACT[self.gain]
```

```
        scale_it = 100.0 / float(self.it_ms)
        return float(raw) * _BASE_LSB_LUX * scale_it / gf

    def read_lux(self):
        if not self.present:
            return None
        try:
            raw = self._read16le(_REG_ALS)
            return self._lux_from_raw(raw)
        except Exception:
            return None

    def read(self):
        if not self.present:
            return (None, None, None)
        try:
            raw = self._read16le(_REG_ALS)
            lux = self._lux_from_raw(raw)
            saturated = (raw >= 0xFF00)  # umbral simple
            return (lux, raw, saturated)
        except Exception:
            return (None, None, None)
```

Listing A.3: `veml7700.py` — Driver VEML7700 robusto

Escaneo I²C dual + prueba VEML7700

```
# scan_i2c_dual.py
from machine import Pin, I2C
import time

def escanear(i2c_id, sda_pin, scl_pin, freq=100_000):
    i2c = I2C(i2c_id, sda=Pin(sda_pin), scl=Pin(scl_pin), freq=freq)
    time.sleep_ms(200)
    devs = []
```

```
    try:
        devs = i2c.scan()
    except Exception as e:
        print(f"I2C{i2c_id} error:", e)
    print(f"I2C{i2c_id} (SDA=GP{sda_pin}, SCL=GP{scl_pin}, {freq} Hz
    ) -> {[hex(d) for d in devs]}")
    return set(devs)

d0 = escanear(0, 4, 5)    # VEML7700
d1 = escanear(1, 10, 11)  # INA260
print("\nResumen esperado:")
print(" - I2C0 deber a contener 0x10 (VEML7700)")
print(" - I2C1 deber a contener 0x40 (INA260)")

from machine import I2C, Pin
import time
from veml7700 import VEML7700

i2c0 = I2C(0, sda=Pin(4), scl=Pin(5), freq=100_000)
print("Scan:", [hex(x) for x in i2c0.scan()])
v = VEML7700(i2c0)

# Barrido simple y ESPERA >= IT
for it in (100, 200, 400):
    v.set_integration(it)
    v.set_gain("2x")
    time.sleep_ms(int(it*1.2))    # espera primera conversi n
    for _ in range(6):
        lux, raw, sat = v.read()
        print(f"IT={it}ms G=2x  raw={raw} sat={sat} lux={lux}")
        time.sleep_ms(int(it*1.2))  # espera nueva muestra
```

Listing A.4: `scan_i2c_dual.py` — Escaneo en I2C0/I2C1 y prueba de lectura VEML

A.1.1. Gestor de sensores (Pico W)

```
# sensor_manager.py
# ------------------------------------------------------
# Gestor de sensores para tu nodo Pico W:
#   - VEML7700 (lux/irradiancia) en I2C0 (SDA=GP4, SCL=GP5)
#   - INA260   (V/I)             en I2C1 (SDA=GP26, SCL=GP27)
#   - (Opcional) temperatura/humedad simulada para pruebas r pidas
#
# Devuelve lecturas como diccionario y permite acumularlas
# en memoria para guardarlas en JSON.
# ------------------------------------------------------

import json
import time
import machine
import random

from veml7700 import VEML7700
from ina260 import INA260

class SensorManager:
    def __init__(self, avg_samples=1):
        """
        avg_samples: cantidad de muestras por lectura para promediar
    (>=1)
        """
        self.data = []
        self.avg_samples = max(1, int(avg_samples))

        # --- I2C0: VEML7700 (lux) ---
        # Pines Pico W: SDA=GP4, SCL=GP5
        self.i2c_veml = machine.I2C(0, sda=machine.Pin(4), scl=
   machine.Pin(5), freq=400_000)
        self.veml = VEML7700(self.i2c_veml)
```

```
32
33          # --- I2C1: INA260 (V/I) ---
34          # Pines Pico W: SDA=GP26, SCL=GP27
35          self.i2c_ina = machine.I2C(1, sda=machine.Pin(26), scl=
    machine.Pin(27), freq=400_000)
36          self.ina = INA260(self.i2c_ina)  # addr por defecto 0x40
37
38      # ---------- Helpers internos ----------
39
40      def _read_veml_lux(self):
41          """Lee lux desde VEML7700 con promedio opcional."""
42          acc = 0.0
43          ok = 0
44          for _ in range(self.avg_samples):
45              try:
46                  acc += float(self.veml.read_lux())
47                  ok += 1
48                  time.sleep_ms(10)
49              except Exception as e:
50                  # Si falla una muestra, seguimos intentando
51                  pass
52          if ok == 0:
53              raise OSError("VEML7700 no responde")
54          return acc / ok
55
56      def _read_ina_voltage(self):
57          """Compatibilidad con tus dos versiones de driver (
    read_voltage/voltage)."""
58          try:
59              return float(self.ina.read_voltage())  # tu versi n
    antigua
60          except AttributeError:
61              return float(self.ina.voltage())       # versi n
    corregida
```

```
62
63      def _read_ina_current(self):
64          """Compatibilidad con tus dos versiones de driver (
    read_current/current)."""
65          try:
66              return float(self.ina.read_current())  # tu versi n
    antigua
67          except AttributeError:
68              return float(self.ina.current())       # versi n
    corregida
69
70      # ---------- API p blica ----------
71
72      def read_sensor(self, simulate_temp_hum=True):
73          """
74          Lee datos de VEML7700 (lux) e INA260 (V, I).
75          Puede simular T/H si  a  n  no integras DHT/AHT.
76
77          Devuelve:
78          {
79            "s": timestamp UNIX,
80            "lux": <float>,
81            "t": <float>,          # temperatura [ C ] (simulada si
    simulate_temp_hum=True)
82            "h": <float>,          # humedad relativa [%] (simulada)
83            "v": <float>,          # voltaje [V]
84            "i": <float>           # corriente [A]
85          }
86          """
87          try:
88              ts = int(time.time())
89
90              # --- Lecturas reales ---
91              iluminancia_lux = round(self._read_veml_lux(), 2)
```

```
92
93              # Validaci n simple (lux no puede ser negativo)
94              if iluminancia_lux < 0:
95                  print("[WARN] Lectura VEML negativa; ajustando a 0")
96                  iluminancia_lux = 0.0
97
98              voltaje_v = round(self._read_ina_voltage(), 3)
99              corriente_a = round(self._read_ina_current(), 3)
100
101             # --- Temperatura/Humedad ---
102             if simulate_temp_hum:
103                 # Simulaci n controlada para pruebas
104                 temperatura_c = round(random.uniform(15.0, 30.0), 1)
105                 humedad_rel = round(random.uniform(35.0, 75.0), 1)
106             else:
107                 # Integrar DHT22/AHT10 reales aqu  si aplica
108                 temperatura_c = None
109                 humedad_rel = None
110
111             return {
112                 "s": ts,
113                 "lux": iluminancia_lux,
114                 "t": temperatura_c,
115                 "h": humedad_rel,
116                 "v": voltaje_v,
117                 "i": corriente_a
118             }
119
120         except Exception as e:
121             print("[ERR] Error leyendo sensores:", e)
122             return None
123
124     def add_measurement(self, simulate_temp_hum=True):
125         """Realiza una lectura y la agrega a self.data si es v lida
```

```
    ."""
126         m = self.read_sensor(simulate_temp_hum=simulate_temp_hum)
127         if m is not None:
128             self.data.append(m)
129             print("[OK] Medici n agregada:", m)
130             return True
131         return False
132 
133     def save_json(self, file_path="data.json"):
134         """Guarda todas las mediciones acumuladas en un archivo JSON
    ."""
135         try:
136             with open(file_path, "w") as f:
137                 json.dump(self.data, f)
138             print("[SAVE] Datos guardados en", file_path)
139         except Exception as e:
140             print("[ERR] Error guardando JSON:", e)
141 
142     def clear_data(self):
143         """Limpia el buffer de mediciones."""
144         self.data = []
145         print("[CLEAR] Buffer de datos limpiado.")
146 
147     # Utilidad: empaquetar una medici n como l nea JSON (para UART
    /LoRa)
148     def read_and_pack(self, simulate_temp_hum=True):
149         """
150         Devuelve una l nea JSON lista para enviar por UART (o
    retornar None si falla).
151         """
152         m = self.read_sensor(simulate_temp_hum=simulate_temp_hum)
153         if m is None:
154             return None
155         try:
```

```
            return json.dumps(m)
        except Exception as e:
            print("[ERR] Error serializando JSON:", e)
            return None
```

Listing A.5: `sensor_manager.py` — Lectura VEML7700/INA260 y empaquetado JSON

A.2. B. Códigos de transmisión y recepción Heltec LoRa

Transmisor (TX)

```
    // ======================== TX: Heltec WiFi LoRa 32 V2
   ========================
#include <Arduino.h>
#ifdef BAND
#undef BAND
#endif
#include "heltec.h"
#include <LoRa.h>
#include <ArduinoJson.h>

// ================ LoRa config ================
static const long    LORA_FREQ       = 915E6;
static const uint8_t LORA_SF         = 9;
static const long    LORA_BW         = 125E3;
static const uint8_t LORA_CR         = 5;        // CR 4/5
static const uint8_t LORA_SYNC_WORD  = 0x34;
static const uint8_t LORA_PREAMBLE   = 12;
static const bool    USE_OLED        = true;
static const int8_t  LORA_TX_POWER   = 14;       // dBm (PA_BOOST)

```

```
20 // ============== App pacing ====================
21 static const uint32_t SEND_PERIOD_MS = 10000;    // 10 s
22 static uint32_t lastSendMs = 0;
23 static uint16_t seq = 0;
24 static uint8_t  sendCount = 0;
25 static const uint8_t MAX_SENDS = 12;             // enviar solo 12
       paquetes
26
27 // ===== Fragmentation layer =====
28 static const uint8_t  FRAG_VER        = 1;
29 static const uint8_t  MAGIC1          = 0xA5;
30 static const uint8_t  MAGIC2          = 0x5A;
31 static const uint16_t CHUNK_BODY_MAX  = 200;     // bytes de payload
       por parte
32 static const uint8_t  CHUNK_HDR_BYTES = 9;       // cabecera fija
33
34 // ========= Utilidades =========
35 void loraConfig() {
36   LoRa.setSpreadingFactor(LORA_SF);
37   LoRa.setSignalBandwidth(LORA_BW);
38   LoRa.setCodingRate4(LORA_CR);
39   LoRa.setSyncWord(LORA_SYNC_WORD);
40   LoRa.setPreambleLength(LORA_PREAMBLE);
41   LoRa.enableCrc();
42   // Heltec fork: firma de 2 par metros
43   LoRa.setTxPower(LORA_TX_POWER, RF_PACONFIG_PASELECT_PABOOST);
44 }
45
46 void oledSplashTX() {
47   if (!USE_OLED) return;
48   Heltec.display->clear();
49   Heltec.display->drawString(0,0,"TX LoRa 915 MsgPack");
50   Heltec.display->drawString(0,10,"SF9/BW125/CR45");
51   Heltec.display->drawString(0,20,"per=10s, n=12");
```

```
52   Heltec.display->display();
53 }
54
55 void dumpHexHead(const uint8_t* p, size_t n, size_t maxShow=32) {
56   Serial.print("[TX] msgpack HEX (");
57   Serial.print(n); Serial.print(" B): ");
58   size_t m = (n<maxShow)?n:maxShow;
59   for (size_t i=0;i<m;i++){
60     char buf[4];
61     sprintf(buf,"%02X ", p[i]);
62     Serial.print(buf);
63   }
64   if (n>maxShow) Serial.print("...");
65   Serial.println();
66 }
67
68 // Crea un JSON con variaciones predefinidas seg n 'k' (0..11) y lo
       serializa a MsgPack
69 // Salida: buffer mp[] y longitud len
70 bool buildPreformattedMsg(uint8_t *mp, size_t mpCap, size_t &len,
      uint8_t k) {
71   StaticJsonDocument<2048> doc;
72
73   // Valores base
74   float baseV  = 1.58f;
75   float baseI  = 0.33f;
76   float baseP  = 0.53f;
77   float baseRH = 47.1f;
78   float baseT  = 21.2f;
79   float baseLux= 120794.1f;
80
81   // Peque as variaciones por  ndice  k
82   float dv  =  0.005f * k;
83   float dI  =  0.002f * k;
```

```
84  float dP  =  0.004f * k;
85  float dRH =  0.3f   * k;
86  float dT  =  0.1f   * k;
87  float dLx =  50.0f  * k;
88
89  doc["node_id"]      = "PICO001";
90  doc["RH_pct"]       = baseRH + dRH;
91  doc["P_W"]          = baseP  + dP;
92  doc["I_A"]          = baseI  + dI;
93  doc["R_load_ohm"]   = 4.7;
94  doc["Pchip_W"]      = 0.0;
95  doc["qf"]           = 1;
96  doc["shade_pct"]    = 0;
97  doc["T_C"]          = baseT + dT;
98  doc["lux"]          = baseLux + dLx;
99  doc["ina_present"]  = 1;
100 doc["seq"]          = (int)seq;    // seq del TX
101 doc["V_V"]          = baseV + dv;
102 doc["cond"]         = "A_ambiente";
103 doc["angle_deg"]    = nullptr;
104 doc["schema"]       = 1;
105 doc["mode"]         = "E1";
106 doc["ts"]           = (long)(millis()/1000UL + 1757000000UL); //
     sin separadores
107 doc["Vbus_meas_V"]  = 0.0;
108 doc["fw_ver"]       = "1.1.0";
109 JsonArray flags = doc.createNestedArray("flags");
110 if (k % 3 == 0) flags.add("vbus_low");
111 if (k % 2 == 0) flags.add("lux_sat");
112 doc["lux_raw"]      = 65535;
113 doc["notes"]        = "";
114
115 len = serializeMsgPack(doc, mp, mpCap);
116 if (len == 0) {
```

```
117     Serial.println("[TX] MsgPack serialize FAILED (buffer too small
        ?)");
118     return false;
119   }
120   dumpHexHead(mp, len);
121   return true;
122 }
123
124 // Env a un buffer grande en partes con cabecera (MAGIC, ver, seq,
        part, total, body_len)
125 void sendFragmented(const uint8_t* buf, size_t len, uint16_t seqVal)
        {
126   const uint8_t parts_total = (len + CHUNK_BODY_MAX - 1) /
        CHUNK_BODY_MAX;
127   for (uint8_t part = 0; part < parts_total; part++) {
128     size_t offset = part * CHUNK_BODY_MAX;
129     size_t chunk  = min((size_t)CHUNK_BODY_MAX, len - offset);
130
131     LoRa.idle();               // garantizar salir de RX
132     LoRa.beginPacket();
133     LoRa.write(MAGIC1);
134     LoRa.write(MAGIC2);
135     LoRa.write(FRAG_VER);
136     LoRa.write((uint8_t)(seqVal & 0xFF));
137     LoRa.write((uint8_t)((seqVal>>8) & 0xFF));
138     LoRa.write(part);
139     LoRa.write(parts_total);
140     LoRa.write((uint8_t)(chunk & 0xFF));
141     LoRa.write((uint8_t)((chunk>>8) & 0xFF));
142     LoRa.write(&buf[offset], chunk);
143     LoRa.endPacket();          // bloqueante
144
145     Serial.print("[TX] part ");
146     Serial.print((int)part+1); Serial.print("/");
```

```
147     Serial.print((int)parts_total);
148     Serial.print(" sent (");
149     Serial.print(chunk); Serial.println(" B)");
150     delay(500); // peque o gap entre partes
151   }
152 }
153
154 void setup() {
155   Serial.begin(115200);
156   delay(100);
157
158   Heltec.begin(USE_OLED, /*LoRa*/true, /*Serial*/true, /*PABOOST*/
       true, LORA_FREQ);
159   loraConfig();
160
161   if (USE_OLED) {
162     Heltec.display->init();
163     Heltec.display->flipScreenVertically();
164   }
165   oledSplashTX();
166
167   Serial.println("LoRa Initial success!");
168   Serial.println("[TX] init OK");
169 }
170
171 void loop() {
172   if (sendCount >= MAX_SENDS) {
173     // Finalizar: no seguimos enviando (queda mostrando "DONE")
174     static bool shown = false;
175     if (!shown && USE_OLED) {
176       Heltec.display->clear();
177       Heltec.display->drawString(0,0, "TX DONE (12)");
178       Heltec.display->drawString(0,10," ltimo  seq="+String(seq-1));
179       Heltec.display->display();
```

```
180       shown = true;
181     }
182     delay(1000);
183     return;
184   }
185 
186   uint32_t now = millis();
187   if (now - lastSendMs < SEND_PERIOD_MS) return;
188   lastSendMs = now;
189 
190   // Construir payload preformateado
191   uint8_t mp[512];
192   size_t  mlen = 0;
193   if (!buildPreformattedMsg(mp, sizeof(mp), mlen, sendCount)) {
194     return;
195   }
196 
197   // Enviar fragmentado
198   sendFragmented(mp, mlen, seq);
199 
200   Serial.print("[TX] sent MsgPack bytes=");
201   Serial.print(mlen);
202   Serial.print(" seq=");
203   Serial.println(seq);
204 
205   if (USE_OLED) {
206     Heltec.display->clear();
207     Heltec.display->drawString(0,0, "TX sent #"+String(sendCount+1)
    +"/12");
208     Heltec.display->drawString(0,10,"seq="+String(seq));
209     Heltec.display->drawString(0,20,"size="+String((int)mlen)+"B");
210     Heltec.display->display();
211   }
212 
```

```
  seq++;
  sendCount++;
}
```

Receptor (RX)

```
    // ======================== RX: Heltec WiFi LoRa 32 V2
    ========================
#include <Arduino.h>
#ifdef BAND
#undef BAND
#endif
#include "heltec.h"
#include <LoRa.h>
#include <ArduinoJson.h>

// ================= LoRa config =================
static const long    LORA_FREQ       = 915E6;
static const uint8_t LORA_SF         = 9;
static const long    LORA_BW         = 125E3;
static const uint8_t LORA_CR         = 5;        // CR 4/5
static const uint8_t LORA_SYNC_WORD  = 0x34;
static const uint8_t LORA_PREAMBLE   = 12;
static const bool    USE_OLED        = true;

// ===== Fragmentation layer =====
static const uint8_t  FRAG_VER        = 1;
static const uint8_t  MAGIC1          = 0xA5;
static const uint8_t  MAGIC2          = 0x5A;
static const uint16_t CHUNK_BODY_MAX  = 200;
static const uint8_t  CHUNK_HDR_BYTES = 9;
static const uint8_t  MAX_PARTS       = 16;
static const uint32_t ASM_TIMEOUT_MS  = 8000;    // holgado para
    terreno
```

```
27
28 // Ensamblador
29 static uint16_t asm_seq = 0xFFFF;
30 static uint8_t  asm_parts_total = 0;
31 static uint16_t asm_part_len[MAX_PARTS];
32 static uint8_t  asm_got_mask = 0;
33 static uint32_t asm_last_time = 0;
34 static uint8_t  asmbuf[1024];
35
36 // Estad sticas
37 static uint32_t stat_parts = 0;
38 static uint32_t stat_ok    = 0;      // mensajes completos OK
39 static uint32_t stat_dup   = 0;      // mensajes completos
     duplicados
40 static uint32_t stat_lost  = 0;      // estimados por saltos de seq
41 static int      last_rssi  = 0;
42 static float    last_snr   = 0;
43 static int32_t  last_seq_full = -1;  //  ltimo  seq completo
44
45 void loraConfig() {
46   LoRa.setSpreadingFactor(LORA_SF);
47   LoRa.setSignalBandwidth(LORA_BW);
48   LoRa.setCodingRate4(LORA_CR);
49   LoRa.setSyncWord(LORA_SYNC_WORD);
50   LoRa.setPreambleLength(LORA_PREAMBLE);
51   LoRa.enableCrc();
52 }
53
54 void oledSplashRX() {
55   if (!USE_OLED) return;
56   Heltec.display->clear();
57   Heltec.display->drawString(0,0,"RX LoRa 915 MsgPack");
58   Heltec.display->drawString(0,10,"SF9/BW125/CR45");
59   Heltec.display->drawString(0,20,"modo terreno");
```

```
60   Heltec.display->display();
61 }
62
63 void asmReset() {
64   asm_seq = 0xFFFF;
65   asm_parts_total = 0;
66   asm_got_mask = 0;
67   memset(asm_part_len, 0, sizeof(asm_part_len));
68 }
69
70 bool asmTimeout() {
71   return (asm_seq != 0xFFFF) && (millis() - asm_last_time >
      ASM_TIMEOUT_MS);
72 }
73
74 void printStats() {
75   float prr = (stat_ok + stat_lost) ? (100.0f * (float)stat_ok / (
      float)(stat_ok + stat_lost)) : 0.0f;
76   Serial.print("[RX] STAT ok=");  Serial.print(stat_ok);
77   Serial.print(" lost=");         Serial.print(stat_lost);
78   Serial.print(" dup=");          Serial.print(stat_dup);
79   Serial.print(" parts=");        Serial.print(stat_parts);
80   Serial.print(" PRR=");          Serial.print(prr,1); Serial.print
      ("%");
81   Serial.print(" last_seq=");     Serial.print(last_seq_full);
82   Serial.print(" last RSSI=");    Serial.print(last_rssi);
83   Serial.print(" SNR=");          Serial.println(last_snr,1);
84 }
85
86 void handleSerialCmd() {
87   // No bloquea; s l o mira si hay un byte disponible.
88   if (!Serial.available()) return;
89   int c = Serial.read();
90   if (c == 'S' || c == 's') {
```

```
    printStats();
  } else if (c == 'R' || c == 'r') {
    stat_ok = stat_dup = stat_lost = stat_parts = 0;
    last_seq_full = -1;
    Serial.println("[RX] STAT reset");
  } else if (c == 'X' || c == 'x') {
    asmReset();
    Serial.println("[RX] ASM cleared");
  }
}

void setup() {
  Serial.begin(115200);
  delay(100);

  Heltec.begin(USE_OLED, /*LoRa*/true, /*Serial*/true, /*PABOOST*/
   true, LORA_FREQ);
  loraConfig();
  LoRa.receive();

  if (USE_OLED) {
    Heltec.display->init();
    Heltec.display->flipScreenVertically();
  }
  oledSplashRX();

  Serial.println("LoRa Initial success!");
  Serial.println("[RX] init OK");
  asmReset();
}

void loop() {
  handleSerialCmd();

```

```
124 if (asmTimeout()) {
125   Serial.println("[RX] asm timeout -> reset");
126   asmReset();
127 }
128
129 int packetSize = LoRa.parsePacket();
130 if (!packetSize) return;
131
132 // Leer paquete entrante
133 uint8_t pkt[256]; int n=0;
134 while (LoRa.available() && n<(int)sizeof(pkt)) {
135   int c = LoRa.read(); if (c<0) break;
136   pkt[n++] = (uint8_t)c;
137 }
138
139 last_rssi = LoRa.packetRssi();
140 last_snr  = LoRa.packetSnr();
141
142 if (n < (int)CHUNK_HDR_BYTES) { Serial.println("[RX] too short");
     return; }
143
144 int idx=0;
145 if (pkt[idx++] != MAGIC1 || pkt[idx++] != MAGIC2) { Serial.println
     ("[RX] bad magic"); return; }
146 uint8_t ver = pkt[idx++]; if (ver != FRAG_VER) { Serial.println("[
     RX] bad ver"); return; }
147 uint16_t seq = (uint16_t)pkt[idx] | ((uint16_t)pkt[idx+1]<<8); idx
     +=2;
148 uint8_t part = pkt[idx++], parts_total = pkt[idx++];
149 uint16_t body_len = (uint16_t)pkt[idx] | ((uint16_t)pkt[idx+1]<<8)
     ; idx+=2;
150
151 if (parts_total==0 || parts_total>MAX_PARTS) { Serial.println("[RX
     ] parts_total invalid"); return; }
```

```
152 if (body_len>CHUNK_BODY_MAX || idx+body_len>n) { Serial.println("[
     RX] body_len invalid"); return; }
153
154 // Ventana por seq
155 if (asm_seq==0xFFFF || seq!=asm_seq) {
156   asmReset();
157   asm_seq = seq;
158   asm_parts_total = parts_total;
159 }
160 if (seq!=asm_seq || part>=asm_parts_total) { Serial.println("[RX]
     out-of-window"); return; }
161
162 // Copiar parte
163 size_t offset = (size_t)part * CHUNK_BODY_MAX;
164 if (offset+body_len > sizeof(asmbuf)) { Serial.println("[RX]
     asmbuf overflow"); asmReset(); return; }
165 memcpy(&asmbuf[offset], &pkt[idx], body_len);
166 asm_part_len[part] = body_len;
167 asm_got_mask |= (1u<<part);
168 asm_last_time = millis();
169 stat_parts++;
170
171 Serial.print("[RX] got part "); Serial.print((int)part+1);
172 Serial.print("/"); Serial.print((int)asm_parts_total);
173 Serial.print(" seq="); Serial.print((int)asm_seq);
174 Serial.print(" rssi="); Serial.print(last_rssi);
175 Serial.print(" snr=");  Serial.println(last_snr,1);
176
177 //  Mensaje  completo?
178 uint8_t want_mask = (uint8_t)((1u<<asm_parts_total)-1u);
179 if (asm_got_mask == want_mask) {
180   size_t total=0; for (uint8_t p=0;p<asm_parts_total;p++) total +=
       asm_part_len[p];
181
```

```
182     // Decodificar MsgPack     JSON
183     StaticJsonDocument<2048> doc;
184     DeserializationError derr = deserializeMsgPack(doc, asmbuf,
    total);
185     if (derr) {
186       Serial.print("[RX] MsgPack decode error: "); Serial.println(
    derr.c_str());
187       asmReset();
188       return;
189     }
190
191     int js_seq = doc["seq"] | -1;
192     bool isDup = false;
193     if (last_seq_full >= 0) {
194       if (js_seq == last_seq_full) {
195         stat_dup++; isDup = true;
196       } else if (js_seq > last_seq_full) {
197         stat_lost += (uint32_t)(js_seq - last_seq_full - 1);
198       }
199     }
200     last_seq_full = js_seq;
201     if (!isDup) stat_ok++;
202
203     float prr = (stat_ok + stat_lost) ? (100.0f * (float)stat_ok / (
    float)(stat_ok + stat_lost)) : 0.0f;
204
205     Serial.print("[RX] JSON decoded ("); Serial.print(total); Serial
    .println(" B):");
206     serializeJsonPretty(doc, Serial); Serial.println();
207
208     Serial.print("[RX] STAT ok=");  Serial.print(stat_ok);
209     Serial.print(" lost=");         Serial.print(stat_lost);
210     Serial.print(" dup=");          Serial.print(stat_dup);
211     Serial.print(" parts=");        Serial.print(stat_parts);
```

```
212     Serial.print(" PRR=");          Serial.print(prr,1); Serial.
    print("%");
213     Serial.print(" last_seq=");     Serial.print(js_seq);
214     Serial.print(" last RSSI=");    Serial.print(last_rssi);
215     Serial.print(" SNR=");          Serial.println(last_snr,1);
216
217     if (USE_OLED) {
218       Heltec.display->clear();
219       Heltec.display->drawString(0,0,  "ok="+String(stat_ok)+" dup
    ="+String(stat_dup));
220       Heltec.display->drawString(0,10, "lost="+String(stat_lost)+"
    PRR="+String(prr,0)+"%");
221       Heltec.display->drawString(0,20, "seq="+String(js_seq));
222       Heltec.display->drawString(0,30, "RSSI "+String(last_rssi)+"
    SNR "+String(last_snr,1));
223       Heltec.display->display();
224     }
225
226     asmReset(); // listo para siguiente mensaje
227   }
228
229   LoRa.receive(); // seguir escuchando
230 }
```

A.3. C.1 Códigos de simulación Escenario 2(CupCarbon)

Transmisor (TX)

```
1
2
3 //Transmisor
4
```

```
//Contador de mensajes eviados
int num 0

//Se demora 5 segundos en empezar el loop para transmitir
delay 5000

//Empieza el loop de transmision
loop
  //Obtiene la posicion del nodo con la estructura lat#long
  getpos pos

  //Obtiene el id del nodo
  atget id id

  //Estructura el mensaje a enviar como lat#long#id#num
  data data pos id num

  //Envia el mensaje
  send data

  //Incrementa el contador de mensajes
  inc num

  //Si ya paso un hora, se detiene la simulacion
  if (num==9600)
    stop
  end

  //Espera 2 segundos para transmitir de nuevo
  delay 5000
```

Listing A.6: Transmisor — envía cada 2 s la trama ‘latlongidnum‘

Receptor (RX)

```
// ====================
// RECEPTOR (gateway m vil)
// ====================

//Receptor

//Empieza loop para recibir mensajes

loop
  //Recibe mensaje, si es que hay
  receive v

  //Transforma mensaje lat#long#id#num a vectores
  vdata data v

  //Asigna cada vector a una variable
  vget lat data 0
  vget long data 1
  vget id data 2
  vget count data 3

  //Anade una fila a archivo log
  printfile lat "   ,   "long "   ,   "id "   ,   "count
```

Listing A.7: Receptor — registra en CSV ‘lat,long,id,count‘

A.4. D.1 Alarmas Thingspeak

```

% Store the channel ID for the moisture sensor channel.
channelID = 2949243;
% Provide the ThingSpeak alerts API key.  All alerts API keys start
   with TAK.
alertApiKey = 'TAK4yh6k+5GbQ1E0lkv';
```

```
% Set the address for the HTTTP call
alertUrl="https://api.thingspeak.com/alerts/send";
% webwrite uses weboptions to add required headers.  Alerts needs a ThingSpeak-Alerts-API-Key header.
options = weboptions("HeaderFields", ["ThingSpeak-Alerts-API-Key", alertApiKey ]);
% Add this line before thingSpeakRead()
readApiKey = '7VHWU4MFNZADM6RJ'; % Find this in your channel's "API Keys" tab


% Thresholds (Modificar para tunear alarmas)
HIGH_IRRADIANCE = 800; % Lux, threshold for high irradiance
LOW_IRRADIANCE = 200;  % Lux, threshold for low irradiance
HIGH_TEMP = 45;        %  C , threshold for high temperature
LOW_TEMP = 10;         %  C , threshold for low temperature
HIGH_HUMIDITY = 80;    % %, threshold for high humidity
LOW_VOLTAGE = 10;      % V, threshold for low voltage
HIGH_VOLTAGE = 20;     % V, threshold for high voltage
LOW_CURRENT = 0.05;    % A, threshold for low current
HIGH_CURRENT = 5;      % A, threshold for high current

% Lectura  ltimo  dato
lastCurrent  = thingSpeakRead(channelID, 'NumPoints', 1, 'Fields', 1, 'ReadKey', readApiKey);
lastVoltage  = thingSpeakRead(channelID, 'NumPoints', 1, 'Fields', 2, 'ReadKey', readApiKey);
lastLux      = thingSpeakRead(channelID, 'NumPoints', 1, 'Fields', 3, 'ReadKey', readApiKey);
lastTemp     = thingSpeakRead(channelID, 'NumPoints', 1, 'Fields', 4, 'ReadKey', readApiKey);
lastHumidity = thingSpeakRead(channelID, 'NumPoints', 1, 'Fields', 5, 'ReadKey', readApiKey);
lastPower = lastVoltage * lastCurrent; % Calcular potencia
```

```
32
33 % Initialize alert variables
34 alertSubject = "";
35 alertBody = "";
36
37
38 % Inicializar lista de alarmas
39 detectedAlarms = {}; %  Aqu  se almacenan todas las alarmas
       detectadas
40
41
42 % ============================================
43 % DETECCI N DE ALARMAS
44 % ============================================
45
46
47 % Check to make sure the data was read correctly from the channel.
48 if isempty(lastCurrent) || isempty(lastVoltage) || isempty(lastLux)
       || isempty(lastTemp)
49     fprintf("Data insuficiente\n");
50 else
51     % 1. ALARMAS POR IRRADIANCIA ALTA (C;V;L)
52     if lastLux > HIGH_IRRADIANCE
53         if lastVoltage < LOW_VOLTAGE && lastCurrent < LOW_CURRENT
54             detectedAlarms{end+1} = "Alta irradiancia + baja
       potencia: Suciedad en paneles, sombreado o c lulas da adas";
55         end
56
57         if lastTemp > HIGH_TEMP && lastPower < (lastVoltage *
       lastCurrent * 0.7)
58             detectedAlarms{end+1} = "Alta irradiancia + alta temp. +
       baja potencia: Sobrecalentamiento reduciendo eficiencia";
59         end
60     end
```

```
61
62      % 2. ALARMAS POR CORRIENTE (C;V;L)
63      if lastCurrent > HIGH_CURRENT && lastVoltage < LOW_VOLTAGE
64          detectedAlarms{end+1} = "Corriente alta + voltaje bajo:
    Cortocircuito parcial o bater a descargada";
65      end
66
67      if lastCurrent < LOW_CURRENT && lastLux > HIGH_IRRADIANCE
68          detectedAlarms{end+1} = "Corriente nula con buena
    irradiancia: Fusible quemado o fallo en inversor";
69      end
70
71      % 3. ALARMAS POR VOLTAJE (C;V;L)
72      if lastVoltage > HIGH_VOLTAGE && lastCurrent < LOW_CURRENT
73          detectedAlarms{end+1} = "Voltaje alto + corriente baja:
    Carga desconectada o bater a cargada";
74      end
75
76      if lastVoltage < LOW_VOLTAGE && lastLux > HIGH_IRRADIANCE
77          detectedAlarms{end+1} = "Voltaje muy bajo + buena
    irradiancia: Fallo de conexi n o c lulas con microfisuras";
78      end
79  end
80
81
82  % Check to make sure the data was read correctly from the channel.
83  if isempty(lastCurrent) || isempty(lastVoltage) || isempty(lastTemp)
84      fprintf("Data insuficiente\n");
85  else
86      % 4. ALARMAS POR TEMPERATURA (C;V;T)
87      if lastTemp > HIGH_TEMP && lastPower < (lastVoltage *
    lastCurrent * 0.8)
88          detectedAlarms{end+1} = "Alta temperatura + baja producci n
    : Efecto t rmico o componente da ado";
```

```
89      end
90  end
91
92
93  % Check to make sure the data was read correctly from the channel.
94  if isempty(lastCurrent) || isempty(lastVoltage) || isempty(
        lastHumidity)
95      fprintf("Data insuficiente\n");
96  else
97      % 5. ALARMAS POR HUMEDAD (C;V;H)
98      if lastHumidity > HIGH_HUMIDITY
99          if lastVoltage < LOW_VOLTAGE * 1.1 || lastCurrent >
        HIGH_CURRENT * 0.9
100             detectedAlarms{end+1} = "Alta humedad + fluctuaciones:
        Problemas de aislamiento o humedad en conexiones";
101         end
102
103         if lastTemp < LOW_TEMP && lastCurrent < LOW_CURRENT
104             detectedAlarms{end+1} = "Alta humedad + baja temp. +
        baja producci n: Condensaci n afectando componentes";
105         end
106     end
107 end
108
109
110
111 % Check to make sure the data was read correctly from the channel.
112 if isempty(lastCurrent) || isempty(lastTemp) || isempty(lastLux)
113     fprintf("Data insuficiente\n");
114 else
115     % 6. ALARMAS POR IRRADIANCIA BAJA (C;T;L)
116     if lastLux < LOW_IRRADIANCE
117         if lastCurrent > HIGH_CURRENT
118             detectedAlarms{end+1} = "Baja irradiancia + alta
```

```
    corriente: Error en sensor o fuga de corriente";
119         end
120
121         if lastTemp > HIGH_TEMP
122             detectedAlarms{end+1} = "Baja irradiancia + alta
    temperatura: Calor por reflejos o acumulaci n  t rmica";
123         end
124     end
125 end
126
127
128 fprintf("EficienSol: %d Fallos detectados en el sistema fotovoltaico
    \n", length(detectedAlarms));
129
130 for i = 1:length(detectedAlarms)
131     fprintf('%s\n', detectedAlarms{i});
132 end
133
134 % Preparar y enviar correo
135 if ~isempty(detectedAlarms)
136     alertSubject = sprintf("EficienSol: %d Fallos detectados en el
    sistema fotovoltaico", length(detectedAlarms));
137     alertBody = strjoin(cellstr(detectedAlarms), newline); % FORZAR
    que sea celda de char
138
139     try
140         webwrite(alertUrl, "body", alertBody, "subject",
    alertSubject, options);
141         fprintf("Correo enviado con %d alarmas.\n", length(
    detectedAlarms));
142     catch someException
143         fprintf("Error al enviar alerta: %s\n", someException.
    message);
144     end
```

```
145 else
146     fprintf("No se detectaron alarmas.\n");
147 end
```

Apéndice B

Read Channel to Trigger Email 1

```
1
2  % Read the soil moisture channel data from the past two weeks.
3  % Send an email and tell the user to add water if the value
4  % is in the lowest 10 %.
5
6  % Store the channel ID for the moisture sensor channel.
7  channelID = 2949243;
8
9  % Provide the ThingSpeak alerts API key.  All alerts API keys start
       with TAK.
10 alertApiKey = <READ_API_KEY>;
11
12 % Set the address for the HTTTP call
13 alertUrl="https://api.thingspeak.com/alerts/send";
14
15 % webwrite uses weboptions to add required headers.  Alerts needs a
       ThingSpeak-Alerts-API-Key header.
16 options = weboptions("HeaderFields", ["ThingSpeak-Alerts-API-Key",
       alertApiKey ]);
17
18 % Set the email subject.
19 alertSubject = sprintf("Probando, Probando");
```

```
20
21 % Add this line before thingSpeakRead()
22 readApiKey = '7VHWU4MFNZADM6RJ'; % Find this in your channel's "API
       Keys" tab
23
24 % Modify the read command to include the key
25 moistureData = thingSpeakRead(channelID,'NumDays',30,'Fields',1,'
       ReadKey',readApiKey);
26
27
28
29 % Check to make sure the data was read correctly from the channel.
30 if isempty(moistureData)
31     alertBody = ' No data read from plant. ';
32     %fprintf("No se envi  Email");
33 else
34     % Calculate a 10% threshold value based on recent data.
35     span = max(moistureData) - min(moistureData);
36     dryValue = 0.1 * span + min(moistureData);
37
38     % Get the most recent point in the array of moisture data.
39     lastValue = moistureData(end);
40
41     % Set the outgoing message
42     if (lastValue<=dryValue)
43         alertBody = ' I need water! ';
44     elseif (lastValue>dryValue)
45         alertBody = ' No water needed. ';
46     end
47
48 end
49
50     % Catch errors so the MATLAB code does not disable a TimeControl
       if it fails
```

```
51      try
52          webwrite(alertUrl , "body", alertBody, "subject",
    alertSubject, options);
53      catch someException
54          fprintf("Failed to send alert: %s\n", someException.message)
    ;
55      end
```

Referencias

[1] Climate Tracker. «Revolución solar en Chile: más sol y menos carbón.» Artículo sobre el crecimiento exponencial de la energía solar en Chile y el potencial del desierto de Atacama. Accedido: 27-08-2025. (2023), dirección: https://climatetrackerlatam.org/historias/revolucion-solar-en-chile-mas-sol-y-menos-carbon-2/ (visitado 27-08-2025).

[2] Generadoras de Chile, *Reporte Anual de Energías Renovables 2023-2024*, Informe gremial con datos de capacidad instalada solar (8 MW en 2013 a más de 9.000 MW en 2023) y récord de 63,8 % de generación renovable. Accedido: 27-08-2025, 2024. dirección: https://generadoras.cl/estudios-y-publicaciones (visitado 27-08-2025).

[3] Comisión Nacional de Energía (Chile), *Reporte Mensual ERNC, vol. 99 – Resumen 2024*, Cifras acumuladas a diciembre 2024: 10.507 MW solares, 47.5 % ERNC. Accedido: 19-08-2025, feb. de 2025. dirección: https://www.cne.cl/wp-content/uploads/2025/02/RMensual_ERNC_v202502.pdf (visitado 19-08-2025).

[4] Ember Climate, *Chile Electricity Data 2024*, Datos de generación eléctrica de Chile. Incluye participación de la energía solar cercana al 20 % en la matriz en

2023. Accedido: 27-08-2025, 2024. dirección: https://ember-energy.org/latest-insights/wind-and-solar-overtake-coal-in-chile/ (visitado 27-08-2025).

[5] Ministerio de Energía de Chile y BMWK (Alemania), *Alianza Energética Chile–Alemania: Potencial renovable y metas al 2030*, Incluye valores de irradiación solar en Atacama y metas al 2030. Accedido: 19-08-2025, mayo de 2022. dirección: https://4echile.cl/wp-content/uploads/2022/01/2022_Factsheet_BMWK_ES_Energy-Partnership.pdf (visitado 19-08-2025).

[6] CORFO, *Solar Industry: Road Map 2016–2025*, Valores de referencia en Atacama: $\sim$2500 kWh/m^2 GHI y $\sim$3500 kWh/m^2 DNI. Accedido: 19-08-2025, 2017. dirección: https://www.corfo.cl/sites/Satellite?blobcol=urldata&blobkey=id&blobtable=MungoBlobs&blobwhere=1475166206777&ssbinary=true (visitado 19-08-2025).

[7] Ministerio de Energía de Chile, *Explorador Solar*, Plataforma oficial de irradiancia solar en Chile. Accedido: 19-08-2025, 2020. dirección: https://solar.minenergia.cl/ (visitado 19-08-2025).

[8] Ministerio de Energía de Chile y Agencia de Sostenibilidad Energética, *Programa Casa Solar*, Programa nacional de cofinanciamiento para sistemas fotovoltaicos de 1–2 kWp en viviendas. Estimación de ahorro: \$150.000 – \$300.000 CLP anuales. Accedido: 19-08-2025, 2024. dirección: https://energia.gob.cl/noticias/nacional/programa-casa-solar (visitado 19-08-2025).

[9] Enel X Chile y Siena Inmobiliaria, *Condominio Canquén Norte: proyecto inmobiliario con autoconsumo solar*, Proyecto residencial en Colina, Santiago, con diseño para autoconsumo solar. Estimación de ahorro:

20 %–30 %. Accedido: 19-08-2025, jul. de 2024. dirección: https://www.enelx.com/cl/es/historias/2024/07/energia-solar-en-sector-inmobiliario (visitado 19-08-2025).

[10] A. Augustin, J. Yi, T. Clausen y W. Townsley, «A Study of LoRa: Long Range & Low Power Networks for the Internet of Things,» *Sensors*, vol. 16, n.º 9, pág. 1466, 2016. DOI: 10.3390/s16091466.

[11] A. N. Utama et al., «Design and Implementation of an Intelligent IoT Platform for Remote Monitoring of Multiple Photovoltaic Plants,» *Sensors*, vol. 23, n.º 15, pág. 6674, 2023, Plataforma IoT multi-planta con dashboards y analítica. DOI: 10.3390/s23156674. dirección: https://www.mdpi.com/1424-8220/23/15/6674 (visitado 28-08-2025).

[12] H.-C. Lee, H.-Y. Liu, T.-C. Lin y C.-Y. Lee, «A Customized Energy Management System for Distributed PV, Energy Storage Units, and Charging Stations on Kinmen Island of Taiwan,» *Energies*, vol. 16, n.º 12, pág. 6331, 2023, Arquitectura de cinco capas, redundancia on-premise/*cloud*. DOI: 10.3390/en16126331. dirección: https://pmc.ncbi.nlm.nih.gov/articles/PMC10256104/ (visitado 28-08-2025).

[13] O. Demir et al., «IoT Based Monitoring and Maintenance of Photovoltaic Systems,» *Applied Sciences*, vol. 13, n.º 24, pág. 13 072, 2023, Arquitectura IoT modular para PV en producción. DOI: 10.3390/app132413072. dirección: https://www.mdpi.com/2076-3417/13/24/13072 (visitado 28-08-2025).

[14] Subsecretaría de Telecomunicaciones de Chile, *Resolución exenta N° 1985, bandas de uso libre en Chile*, Banda de 915 MHz habilitada para IoT en Chile. Accedido: 27-08-2025, 2017. dirección: https://www.subtel.gob.cl.

[15] Z. Qian, D. R. Thinesh y V. Ayyalusamy, «A LoRaWAN Based Solar PV Condition Monitoring System,» en *2022 IEEE International Conference on Power Electronics, Drives and Energy Systems (PEDES)*, Configuración LoRaWAN para monitoreo de condición PV, 2022. DOI: 10.1109/PEDES56012.2022.10080221. dirección: https://sutd-isl.github.io/research/LoRa-PV/ (visitado 28-08-2025).

[16] J. M. Paredes-Parra, A. J. García-Sánchez, A. Mateo-Aroca y Á. Molina-García, «An Alternative Internet-of-Things Solution Based on LoRa for PV Power Plants: Data Monitoring and Management,» *Energies*, vol. 12, n.º 5, pág. 881, 2019. DOI: 10.3390/en12050881.

[17] H.-C. Kim, H.-J. Park y S. H. Choi, «Monitoring a PV Junction Box Based on LoRa,» *Electronics*, vol. 11, n.º 7, pág. 1175, 2022. DOI: 10.3390/electronics11071175.

[18] M. Hameed y H. Kurnaz, «A Secure Low-Cost Photovoltaic Monitoring System Based on LoRaWAN and Artificial Intelligence,» *Discover Computing*, vol. 2, n.º 1, pág. 34, 2024, LoRaWAN + IA; foco en seguridad y bajo costo. DOI: 10.1007/s10791-024-09475-0. dirección: https://link.springer.com/article/10.1007/s10791-024-09475-0 (visitado 28-08-2025).

[19] P. de completar, «Pendiente: Monitoreo de salud a nivel de panel con IoT (WiFi+MQTT),» *Pendiente*, 2021, Placeholder temporal para [19]; reemplazar con referencia exacta (autores, título, revista, DOI/URL).

[20] M. Tradacete-Ágreda, E. Santiso-Gómez et al., «High-Performance IoT Module for Real-Time Control and Self-Diagnose PV Panels under Working Daylight and Dark Electroluminescence Conditions,» *Internet of Things*, vol. 25, pág. 101 006, 2024, Panel-level IoT con I–V, EL y reconfiguración activa. DOI: 10.1016/j.iot.2023.101006. dirección:

https://www.sciencedirect.com/science/article/pii/S2542660523003293 (visitado 28-08-2025).

[21] *ThingSpeak Documentation*, Consultado el 19 de agosto de 2025, 2025. dirección: https://www.mathworks.com/help/thingspeak/ (visitado 19-08-2025).

[22] D. Benavides et al., «Energy Management Model for a Remote Microgrid Based on Demand-Side Energy Control,» *Energies*, vol. 17, n.º 1, pág. 170, 2023, Implementa *ThingSpeak* con canales de PV, clima y demanda. DOI: 10.3390/en17010170. dirección: https://www.mdpi.com/1996-1073/17/1/170 (visitado 28-08-2025).

[23] P. de completar, «Pendiente: Navegación/autonomía de UAV para inspección de pasillos en plantas FV,» *Pendiente*, 2022, Placeholder temporal para [23]; reemplazar con referencia exacta (autores, título, revista, DOI/URL).

[24] P. de completar, «Pendiente: Detección de hot-spots PV con UAV y mapeo georreferenciado multiespectral,» *Pendiente*, 2024, Placeholder temporal para [24]; reemplazar con referencia exacta (autores, título, revista, DOI/URL).

[25] P. de completar, «Pendiente: Estimación de soiling a nivel celda con imágenes RGB (error RMS ¡1 %),» *Pendiente*, 2024, Placeholder temporal para [25]; reemplazar con referencia exacta (autores, título, revista, DOI/URL).

[26] M. S. Ramírez, F. García-Márquez et al., «Convolutional Neural Networks and Internet of Things for Fault Detection by Aerial Monitoring of Photovoltaic Solar Plants,» *Measurement*, vol. 234, pág. 114 861, 2024, CNN + IoT con monitoreo aéreo para fallas en plantas FV. DOI: 10.1016/j.measurement.2024.114861. dirección: https://www.sciencedirect.com/science/article/pii/S0263224124000953 (visitado 28-08-2025).

[27] Q. Guo, T. Sun, X. Liu et al., «Experimental Evaluation of the Packet Reception Performance of LoRa,» *Sensors*, vol. 21, n.º 4, pág. 1071, 2021. DOI: 10.3390/s21041071.