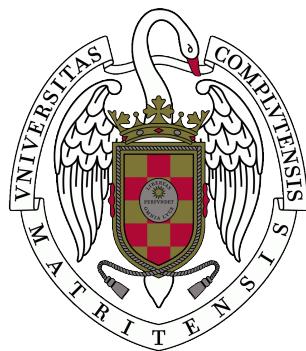


---

**Diseño e implementación de un sistema de adquisición  
transmisión y visualización de datos basado en CanSat**  
**Design and Implementation of a CanSat-Based System  
for Data Acquisition Transmission and Visualization**

---



**Trabajo de Fin de Máster**  
**Curso 2024–2025**

**Autor**  
Sergio García Sánchez

**Director**  
Adrián Riesco Rodríguez

**Máster en Ingeniería Informática**  
**Facultad de Informática**  
**Universidad Complutense de Madrid**



Diseño e implementación de un sistema de adquisición transmisión y visualización de datos basado en CanSat

Design and Implementation of a  
CanSat-Based System for Data Acquisition  
Transmission and Visualization

Trabajo de Fin de Máster en Ingeniería Informática  
Departamento de XXXXXXXXXXXXXXXX

Autor  
Sergio García Sánchez

Director  
Adrián Riesco Rodríguez

Convocatoria: *Febrero/Junio/Septiembre 2025*  
Calificación: *Nota*

Máster en Ingeniería Informática  
Facultad de Informática  
Universidad Complutense de Madrid

**DIA de MES de AÑO**



# Dedicatoria

*A Pedro Pablo y Marco Antonio, por crear TeXiS  
e iluminar nuestro camino*



# Agradecimientos

A Guillermo, por el tiempo empleado en hacer estas plantillas. A Adrián, Enrique y Nacho, por sus comentarios para mejorar lo que hicimos. Y a Narciso, a quien no le ha hecho falta el Anillo Único para coordinarnos a todos.



# Resumen

## **Diseño e implementación de un sistema de adquisición transmisión y visualización de datos basado en CanSat**

Este trabajo tiene como objetivo principal el desarrollo de una plataforma de visualización de datos en tiempo real para proyectos tipo CanSat que permita ser reutilizada en diferentes proyectos con diferente hardware. Para validar la solución propuesta, se ha construido un prototipo funcional basado en un CanSat, que integra diferentes tipos de sensores, módulo GNSS, transmisión por radiofrecuencia, cámara y además se integra con la plataforma.

La plataforma desarrollada se estructura en distintos módulos independientes (adquisición de datos, transmisión, backend, frontend) que pueden adaptarse fácilmente a nuevas configuraciones sin necesidad de modificar el sistema en su conjunto. Este enfoque facilita la modificación del sistema y su integración en otros contextos con requisitos similares.

Durante el desarrollo se han cubierto todas las etapas clave, desde la adquisición y tratamiento de los datos hasta su visualización en tiempo real. Se han realizado pruebas de funcionamiento que demuestran tanto la robustez de la arquitectura como la utilidad práctica del sistema.

## **Palabras clave**

CanSat, Visualización de datos, Arquitectura modular, Plataforma IoT, Telemetría, LoRa, WebSocket



# Abstract

## **Design and Implementation of a CanSat-Based System for Data Acquisition Transmission and Visualization**

An abstract in English, half a page long, including the title in English. Below, a list with no more than 10 keywords.

### **Keywords**

10 keywords max., separated by commas.



# Índice

<b>1. Introducción</b>	<b>1</b>
1.1. Contexto . . . . .	1
1.2. Motivación . . . . .	2
1.3. Objetivos . . . . .	2
1.4. Plan de trabajo . . . . .	3
1.5. Código fuente y licencia . . . . .	4
1.6. Organización de la memoria . . . . .	4
<b>2. Trabajo relacionado</b>	<b>7</b>
2.1. Proyectos educativos y competiciones CanSat . . . . .	7
2.2. Sistemas de adquisición y transmisión de datos para CanSat . . . . .	8
2.3. Soluciones existentes para telemetría y visualización de datos . . . . .	10
<b>3. Fundamentos teóricos</b>	<b>11</b>
3.1. Comparativa de microcontroladores y microcomputadores . . . . .	11
3.2. Interfaces de comunicación serie . . . . .	15
3.3. Tecnologías de comunicación por radiofrecuencia en sistemas embebidos . .	17
3.4. Sensores empleados para telemetría: presión barométrica, IMU y GNSS . .	20
3.5. Captura y transmisión de vídeo en tiempo real . . . . .	22
3.5.1. Captura digital de vídeo . . . . .	23
3.5.2. Protocolos de transmisión de vídeo en tiempo real . . . . .	25
3.6. Visualización de datos en tiempo real: arquitecturas orientadas a eventos .	26
3.6.1. Definición de arquitecturas orientadas a eventos . . . . .	26
3.6.2. Flujo típico de eventos hacia el cliente web . . . . .	26
3.6.3. RabbitMQ . . . . .	27
3.6.4. WebSockets . . . . .	28

<b>4. Diseño e implementación del sistema</b>	<b>29</b>
4.1. Selección de componentes y tecnologías utilizadas . . . . .	29
4.2. Arquitectura general del sistema . . . . .	33
4.3. Montaje electrónico del CanSat . . . . .	35
4.4. Código embebido en la Raspberry Pi . . . . .	42
4.4.1. Captura de vídeo en tiempo real . . . . .	43
4.4.2. Adquisición y envío de telemetría . . . . .	43
4.4.3. Envío por LoRa . . . . .	43
4.4.4. Conexión con RabbitMQ . . . . .	43
4.4.5. Formato de los eventos . . . . .	44
4.5. Software de la estación de tierra . . . . .	44
4.6. Backend con Spring Boot . . . . .	44
4.6.1. Recepción de eventos . . . . .	45
4.6.2. Persistencia de eventos . . . . .	45
4.6.3. Acceso a eventos . . . . .	45
4.6.4. Distribución en tiempo real . . . . .	45
4.7. Frontend con Flutter para visualización en tiempo real . . . . .	45
4.7.1. Arquitectura general . . . . .	46
4.7.2. Visualización modular . . . . .	46
4.7.3. Adaptabilidad . . . . .	48
4.8. Pruebas de integración y validación . . . . .	49
<b>5. Conclusiones y Trabajo Futuro</b>	<b>51</b>
<b>6. Introduction</b>	<b>53</b>
<b>7. Conclusions and Future Work</b>	<b>55</b>
<b>Bibliografía</b>	<b>57</b>
<b>A. Descarga e instalación del proyecto</b>	<b>61</b>

# Índice de figuras

1.1. Diagrama básico de un CanSat. Fuente: ResearchGate (2018) . . . . .	2
2.1. Medidas máximas de un CanSat para una competición europea . . . . .	8
2.2. Ejemplo de comunicación Uplink/Downlink . . . . .	9
3.1. Distribución de pines GPIO en Raspberry Pi Zero 2 . . . . .	12
3.2. Distribución de pines GPIO en ESP32 . . . . .	13
3.3. Distribución de pines en Arduino Nano . . . . .	14
3.4. Configuración SPI con un maestro y un esclavo . . . . .	16
3.5. Configuración I <sup>2</sup> C con un maestro y varios esclavos . . . . .	16
3.6. Ejemplo de conexión UART . . . . .	17
3.7. Sistema de referencia basado en los ángulos de pitch, yaw y roll . . . . .	21
3.8. Ejemplo de cálculo de la posición usando la técnica trilateración . . . . .	22
3.9. Ejemplo de los distintos tipos de exchange de RabbitMQ . . . . .	27
4.1. Raspberry Pi Zero 2 W . . . . .	29
4.2. Sensor de presión barométrica bmp388 . . . . .	30
4.3. IMU bno085 . . . . .	30
4.4. GPS bn-880 . . . . .	31
4.5. Raspberry Pi Camera Module 2 . . . . .	31
4.6. Módulo LoRa E32-900T20D . . . . .	31
4.7. Antena LoRa U.FL a SMA 868–915 MHz, 3 dBi . . . . .	32
4.8. Cargador MCP73871 . . . . .	32
4.9. Batería 18650 . . . . .	33
4.10. Arquitectura general del sistema . . . . .	35
4.11. Diagrama de conexión de los sensores y módulo LoRa . . . . .	36
4.12. Diagrama de conexión del módulo GNSS BN-880 mediante el adaptador CP2102 . . . . .	37

4.13. Diagrama de conexión del cargador MCP73871 y booster . . . . .	38
4.14. Diagrama de conexión de la estación de tierra con Raspberry Pi 4 y módulo LoRa . . . . .	40
4.15. Montaje final del sistema en la PCB (vista frontal) . . . . .	41
4.16. Montaje final del sistema en la PCB (vista trasera) . . . . .	42
4.17. Ejemplo del Componente Metrics . . . . .	46
4.18. Ejemplo del Componente Attitude . . . . .	47
4.19. Ejemplo del Componente TelemetryChart . . . . .	47
4.20. Ejemplo del Componente Map . . . . .	48
4.21. Interfaz en modo escritorio . . . . .	48
A.1. Clonado del repositorio . . . . .	61
A.2. Ejecución del comando docker ps . . . . .	62
A.3. Interfaz Web corriendo en el puerto 80 . . . . .	62
A.4. Interfaz RabbitMQ corriendo en el puerto 15672 . . . . .	63

# Índice de tablas

3.1.	Comparativa de microcontroladores y microcomputadores . . . . .	15
3.2.	Comparativa entre las interfaces SPI, I <sup>2</sup> C y UART . . . . .	18
3.3.	Comparativa de tecnologías de comunicación por radiofrecuencia . . . . .	20
3.4.	Comparativa de sensores de presión barométrica . . . . .	20
3.5.	Comparativa de sensores IMU . . . . .	21
3.6.	Comparativa de las principales constelaciones GNSS . . . . .	22
3.7.	Comparativa de receptores GNSS comunes en sistemas embebidos . . . . .	22
3.8.	Comparativa entre códecs de vídeo comunes . . . . .	24
3.9.	Comparativa de formatos contenedores de vídeo . . . . .	25
3.10.	Comparativa de protocolos de transmisión de vídeo en tiempo real . . . . .	26
4.1.	Resumen de conexiones de los periféricos a la Raspberry Pi Zero 2 W . . . . .	39
4.2.	Conexiones del módulo LoRa a la Raspberry Pi 4 en la estación de tierra . . . . .	40



Capítulo **1**

# Introducción

Este Trabajo de Fin de Máster presenta el diseño e implementación de un sistema completo de adquisición, transmisión y visualización de datos en tiempo real inspirado en el concepto CanSat.

El proyecto está formado por la construcción de un dispositivo tipo CanSat, con diferentes sensores, GNSS, cámara y comunicación por wifi o radio, además, una plataforma web opensource encargada de visualizar los datos recogidos en tiempo real.

Esta plataforma se ha diseñado como una herramienta genérica y reutilizable de forma que se pueda adaptar fácilmente a otros proyectos similares. A lo largo del documento se describen el contexto del trabajo, la motivación, los objetivos planteados, la planificación seguida y la estructura de la memoria.

## 1.1. Contexto

El proyecto CanSat propuesto por el profesor Robert J. Twiggs en 1998 Japan Aerospace Exploration Agency (2003) comienza como un proyecto educativo basado en la simulación de un nanosatélite del tamaño de una lata de refresco y de un peso alrededor de los 350 gramos, el objetivo es ayudar a los alumnos de distintos niveles a entender todas las fases de desarrollo de un satélite, desde la elección de la misión científica hasta la integración del sistema completo, incluyendo los sensores necesarios para obtener los datos necesarios para dicha misión, la electronica necesaria para usar dichos sensores y enviarlos por radio a una estación de tierra, la visualización de los datos, el diseño de una carcasa 3D capaz de aguantar la fuerza del lanzamiento y el diseño de un paracaídas.

Los CanSat no son puestos en órbita, pero son lanzados por cohetes a escala, globos aerostáticos o drones, esto los somete a distintas fuerzas externas como aceleración, vibraciones o posibles impactos, lo que hace que el CanSat tenga que tener una estructura resistente. La misión del CanSat es usar los sensores para recoger datos durante el descenso y transmitirlos a la estación de tierra.

Durante los últimos años la popularidad del proyecto ha ido aumentando, llegando a crear competiciones nacionales e internacionales lideradas por agencias espaciales como la agencia espacial europea European Space Agency (2024) con el objetivo de promover el interés por el sector aeroespacial y por las carreras STEM en general desde pequeños.

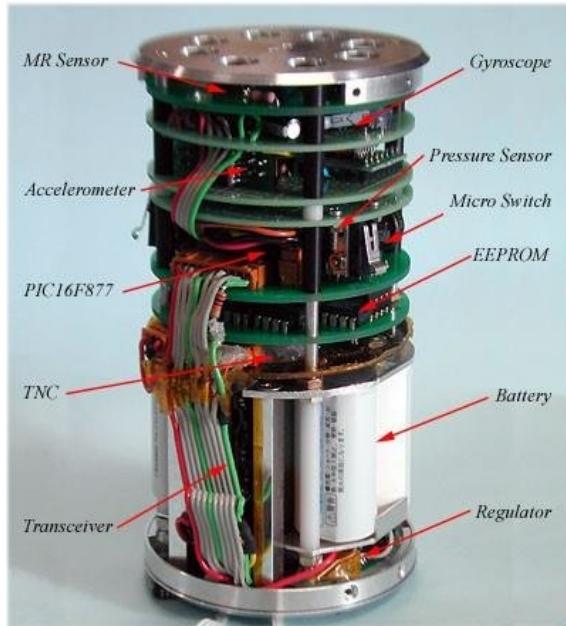


Figura 1.1: Diagrama básico de un CanSat. Fuente: ResearchGate (2018)

## 1.2. Motivación

Debido a que estos proyectos suelen estar más enfocados en la parte electrónica (recogida y transmisión de datos) y no tanto en la visualización de datos, esta última parte suele quedar más descuidada, implementándose solo soluciones básicas como visualización de datos por consola o gráficas simples. Además, estas soluciones son específicas para un CanSat concreto, lo que obliga a implementar soluciones desde cero.

Por ello, surge la necesidad de crear una plataforma común y reutilizable que permita visualizar en tiempo real los datos enviados por el CanSat y recibidos por la antena de manera más visual y profesional sin depender de un hardware concreto. Con la creación de esta plataforma se facilitaría el análisis de los datos durante las pruebas y el lanzamiento, además puede servir como base para futuras implementaciones específicas, ayudando a que los alumnos integren gráficas avanzadas sin tener que desarrollar una plataforma completa.

## 1.3. Objetivos

El objetivo principal de este proyecto consiste en diseñar y desarrollar desde cero un satélite tipo CanSat y la implementación una plataforma de visualización de datos reutilizable. Para ello se han dividido los objetivos en dos partes diferenciadas: investigación de las tecnologías actuales y desarrollo del CanSat.

### ■ Investigación

- Comparación de los distintos microcontroladores y microcomputadores existentes para determinar cuál se ajusta mejor a los objetivos de nuestro desarrollo.
- Estudio de los distintos sensores disponibles en el mercado y su comunicación con el microcomputador.

- Análisis de las diferentes opciones de alimentación para el CanSat y de la posibilidad de cargarse con paneles solares.
- Comparación de las herramientas actuales de visualización de datos.
- **Desarrollo**, dividido en dos partes: la creación del hardware del CanSat y la plataforma de visualización.
  - CanSat
    - Creación de un CanSat que cumpla con las medidas básicas (66mm x 115mm) y peso (entre 300g y 350g).
    - Integración de receptor GPS
    - Cámara para transmisión de video en tiempo real.
    - Sensor de presión, temperatura y altitud.
    - Giroscopio para obtener la orientación del dispositivo.
    - Batería con posibilidad de carga mediante paneles solares.
    - Retransmisión de los datos por WIFI si el dispositivo tiene conexión a internet.
    - Retransmisión de los datos por radio en caso de que no tenga conexión.
    - Desarrollo de un receptor de radio en la estación terrestre.
  - Plataforma de visualización
    - Visualización en tiempo real de toda la telemetría recibida.
    - Visualización del último valor de cada elemento de la telemetría.
    - Gráficas en tiempo real de los valores recibidos.
    - Visualización en tiempo real de las imágenes transmitidas.
    - Mapa con la ubicación exacta del CanSat.
    - Modelo 3D con la orientación real del CanSat.
    - Descarga de los datos de telemetría para un rango de fechas concreto.

## 1.4. Plan de trabajo

Durante el desarrollo del proyecto se ha seguido un plan de trabajo con el objetivo de organizar las tareas y alcanzar los objetivos mencionados anteriormente. Este plan de trabajo se ha dividido en los siguientes puntos:

- Revisión y comparación de los microprocesadores Arduino y ESP32 y del microcomputador Raspberry Pi Zero 2 en función de sus capacidades y compatibilidad con los sensores y módulos de comunicación.
- Selección de los sensores (GPS, altímetro y giroscopio), módulo de comunicación (LoRa), cámara y sistema de alimentación.
- Ensamblaje inicial de los distintos componentes electrónicos usando placas de prototipado, lo que facilita las pruebas individuales de cada componente.
- Definición de la arquitectura general del sistema compuesta por tres componentes principales:

- Código embebido en la Raspberry Pi encargado de leer los sensores y transmisión de datos.
  - Receptor de radio en tierra encargado de la recepción de los datos enviados por el CanSat.
  - Plataforma de visualización en tiempo real.
- Implementación del código embebido en la Raspberry usando Python y las distintas librerías para interactuar con cada sensor.
  - Implementación de la plataforma de visualización en tiempo real basada en eventos, usando un sistema de colas RabbitMQ, un backend en Java y Spring Boot, un frontend en Flutter y una base de datos PostgreSQL para guardar los datos de telemetría.
  - Soldar los componentes electrónicos en una placa definitiva de un tamaño que cumpla con las medidas reglamentarias de CanSat.
  - Realización de pruebas de integración una vez todo el sistema esté terminando, incluyendo pruebas de duración de batería y de alcance de comunicaciones por radio.
  - Redacción de la memoria documentando los pasos seguidos en el proyecto y los resultados obtenidos.

## 1.5. Código fuente y licencia

Todo el código desarrollado para este proyecto se encuentra disponible en un único repositorio público bajo la licencia MIT.

Incluye el backend, frontend y los scripts embebidos tanto en el CanSat como en la estación de tierra.

El repositorio está alojado en GitHub:

<https://github.com/tronxi/real-time-tracking-system>

## 1.6. Organización de la memoria

La memoria se compone de los siguientes capítulos:

- **Capítulo 1. Introducción:** se presenta el contexto del proyecto, la motivación personal y académica, los objetivos planteados y el plan de trabajo seguido.
- **Capítulo 2. Trabajo relacionado:** se revisan iniciativas previas relacionadas con el concepto CanSat, tanto en el ámbito educativo como técnico, y se analizan distintas soluciones de adquisición, transmisión y visualización de datos.
- **Capítulo 3. Fundamentos teóricos:** se definen los conceptos técnicos necesarios para el desarrollo del sistema, incluyendo comparativas de hardware, tecnologías de comunicación, sensores, protocolos de vídeo y arquitecturas de visualización en tiempo real.

- **Capítulo 4. Diseño e implementación del sistema:** se describe la arquitectura del sistema completo, los componentes elegidos, el montaje del CanSat, el desarrollo del software embebido, el backend, el frontend y las pruebas realizadas.
- **Capítulo 5. Conclusiones y trabajo futuro:** se recogen las principales conclusiones del trabajo, se valoran los resultados obtenidos y se proponen posibles mejoras y líneas de desarrollo futuras.



# Capítulo 2

## Trabajo relacionado

En este capítulo vamos a contextualizar el concepto CanSat y se revisarán trabajos relacionados con este tipo de sistemas.

Primero se describirá en detalle que es un CanSat y las principales iniciativas educativas que los promueven, algunas competiciones educativas y normativa para participar.

A continuación, se presentarán las soluciones más habituales de adquisición y transmisión de datos, así como las herramientas existentes para visualización de telemetría en tiempo real.

### 2.1. Proyectos educativos y competiciones CanSat

Desde su creación en 1998 por Robert J. Twiggs, el concepto CanSat ha sido muy utilizado para referirse a satélites suborbitales de bajo coste, ligeros y de tamaño contenido. El objetivo de estos satélites está mayormente asociado al ámbito educativo, donde se usan para dar una introducción a los estudiantes al desarrollo real de una misión espacial, teniendo en cuenta todas las fases de una misión científica real, definición de los objetivos científicos, elección de los componentes, diseño de la arquitectura, ensamblaje de los componentes, diseño de la estación de tierra y visualización y procesado de los datos.

En la actualidad, varias instituciones y agencias espaciales realizan competiciones anuales dirigidas a estudiantes de todas las edades para desarrollar sus propios CanSat. Estas competiciones se basan en lanzar el CanSat con un cohete a escala, desde un drone o globo aerostático y recibir datos durante el tiempo de descenso.

Algunas de estas competiciones son la American CanSat Competition (2025), organizada por la American Astronautical Society (2025) dirigida a estudiantes universitarios y European Cansat Competition (2025) organizada por la European Space Agency (2025) y dirigida a grupos de estudiantes de entre 14 y 19 años. Esta competición organiza torneos regionales y una final a nivel europeo con los mejores representantes de cada país.

Los requisitos que debe cumplir un CanSat para participar en una competición europea son:

- Deben tener una altura máxima de 115mm y un diámetro máximo de 66mm.
- Un peso entre 300 y 350 gramos.

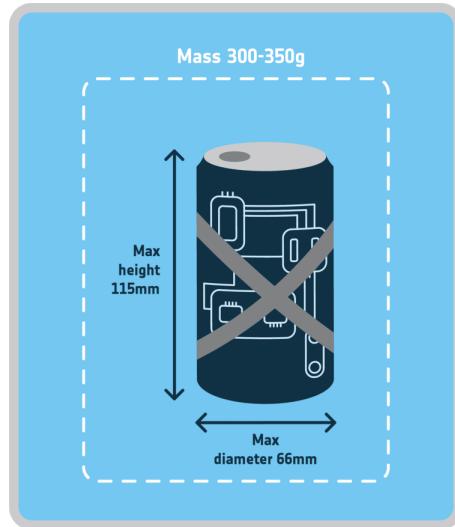


Figura 2.1: Medidas máximas de un CanSat para una competición europea

- Deben cumplir una misión principal basada en medir la presión y temperatura del aire y enviarlo a la estación de tierra al menos una vez por segundo.
- Una misión secundaria elegida por cada equipo.
- No deben incluir materiales inflamables, explosivos o peligrosos para el medio ambiente.
- Debe estar alimentado por batería o paneles solares que lo mantengan encendido durante al menos cuatro horas.
- La batería debe ser accesible y fácil de reemplazar o cargar.
- Incluir un interruptor de encendido y apagado accesible.
- Incluir un paracaídas que facilite la recuperación del CanSat después del lanzamiento y que garantice un tiempo de vuelo máximo de 120 segundos.
- El ratio de descenso debe estar entre 8 y 11m/s.
- Soportar una aceleración de hasta 20g.
- El coste total del CanSat no puede superar los 500€.

## 2.2. Sistemas de adquisición y transmisión de datos para CanSat

Los sistemas de adquisición y transmisión de datos utilizados en los CanSat siguen una arquitectura similar a la empleada en las misiones espaciales reales. Esta arquitectura se basa en dos flujos de comunicación realizados entre el satélite y la estación de tierra:

- **Uplink:** Enviar telecomandos al satélite desde tierra, en el caso de los CanSat esto es poco común.

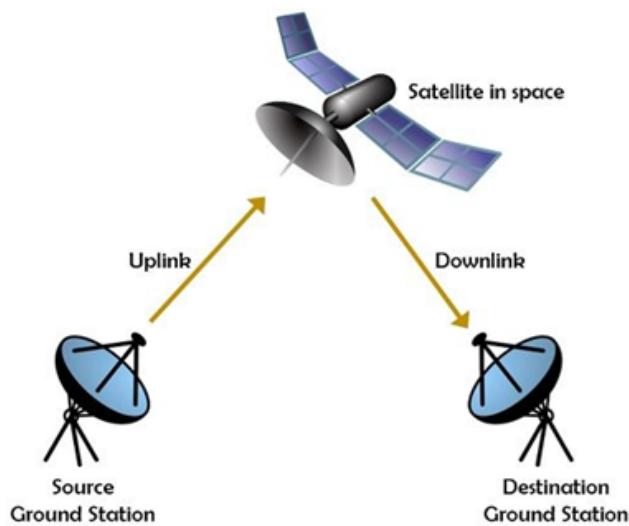


Figura 2.2: Ejemplo de comunicación Uplink/Downlink

- **Downlink:** Recibir telemetría desde el satélite a la estación de tierra.

Para llevar a cabo esta comunicación, los CanSat deben incluir una serie de componentes básicos:

- **Ordenador a bordo (OBC, On-Board Computer):** Encargado de la comunicación con los sensores y el módulo de transmisión. Puede estar basado en microcontroladores (como Arduino o ESP32) o microcomputadores (como Raspberry Pi).
- **Módulo de comunicación:** se encarga de la transmisión de datos mediante tecnologías como LoRa Corporation (2015), XBee International (2024) o Wi-Fi, dependiendo del alcance necesario, la elección de la frecuencia de transmisión depende tanto del módulo que se utilice como de la normativa de cada país, además, debe poderse cambiar con facilidad para no interferir con otros CanSat.
- **Sensores:** permiten adquirir información relevante sobre el entorno, como presión atmosférica, temperatura, orientación (IMU) o localización geográfica (GPS).
- **Fuente de alimentación:** normalmente una batería recargable, capaz de mantener operativo el sistema durante toda la misión. En algunos casos puede complementarse con pequeños paneles solares.

Estos componentes forman la base mínima que un CanSat necesita para poder llevar a cabo la misión científica y comunicar los datos con la estación de tierra.

### 2.3. Soluciones existentes para telemetría y visualización de datos

Normalmente, los proyectos CanSat están más enfocados en el diseño electrónico y la comunicación por radio, por lo que se suele desarrollar menos la parte de visualización de datos, utilizando por lo general herramientas genéricas o limitadas al ordenador en el que está corriendo la estación de tierra, algunas de estas herramientas son:

- **SerialPlot**: herramienta ligera que permite graficar en tiempo real los datos recibidos por puerto serie. Es muy usada durante el desarrollo por su sencillez y rapidez para validar sensores, aunque no permite guardar datos ni personalizar la interfaz.
- **Matplotlib** Hunter (2003), **PyQtGraph** o **Tkinter**: librerías desarrolladas en python, permiten construir interfaces personalizadas, pero no están pensadas para ser accesibles a través de una interfaz web y requieren conocimientos de programación hasta para los casos más simples.
- **Excel** o **Sheets**: se utilizan exportando directamente los datos recibidos, no requieren conocimientos de programación pero no permiten la visualización en tiempo real.
- **LabVIEW National Instruments** entorno gráfico profesional diseñado para adquisición y visualización de datos en sistemas embebidos e instrumentación industrial. Permite construir interfaces complejas mediante programación visual y cuenta con soporte nativo para muchos dispositivos de hardware. Aunque es muy potente, tiene un coste elevado de licencia y una curva de aprendizaje considerable, por lo que raramente se utiliza en proyectos educativos como CanSat.

En resumen, la gran mayoría de herramientas son o muy básicas o demasiado potentes para el uso necesario en un CanSat, además ninguna de estas es específica para el tipo de datos y gráficos habituales en este tipo de proyectos.

# Capítulo 3

## Fundamentos teóricos

Este capítulo se enfoca en analizar en profundidad los aspectos técnicos necesarios para llevar a cabo el desarrollo del CanSat. Se incluye una comparativa entre los distintos microcontroladores y microcomputadores disponibles en el mercado, evaluando su precio, disponibilidad y adecuación a los requisitos técnicos del proyecto. Además, se analizan los sensores y módulos de comunicación más comunes, así como los principales protocolos utilizados para la comunicación entre el microcontrolador y los sensores, como I2C y UART. También se estudian distintas opciones para la retransmisión de vídeo en tiempo real desde el CanSat. Por último, se exploran soluciones para la visualización de los datos a través de una interfaz web en tiempo real.

### 3.1. Comparativa de microcontroladores y microcomputadores

Uno de los componentes principales de un CanSat o de cualquier sistema embebido en general es su microcontrolador o microcomputador, es el encargado de comunicarse con los sensores, procesar los datos y transmitirlo a través del módulo de comunicación, también es el encargado de procesar el video y retransmitirlo en tiempo real(si el hardware lo permite).

Primero conviene aclarar la diferencia entre microcontrolador y microcomputador:

- **Microcontrolador:** Circuito integrado que combina procesador, memoria y periféricos de entrada/salida en un solo chip. Está diseñado para realizar tareas específicas con bajo consumo energético y recursos limitados. Es común en aplicaciones como lectura de sensores, control de motores o gestión de comunicaciones básicas. Ejemplos comunes son Arduino Uno o ESP32
- **Microcomputador:** Sistema completo en una sola placa (Single-Board Computer) que integra procesador, memoria, almacenamiento y puertos de expansión. Es capaz de ejecutar sistemas operativos completos (como Linux) y realizar tareas más complejas, como procesamiento de imágenes, servidor web o interfaces gráficas. Un ejemplo típico es la Raspberry Pi Zero.

Actualmente, en el mercado se pueden encontrar varios de estos microcontroladores o microcomputadores de bajo coste, tamaño y peso reducido y bajo consumo. En esta sección

nos vamos a centrar en el análisis de tres de las opciones más populares:

- **Raspberry Pi Zero 2 W:** La Raspberry Pi Zero 2 es un Single Board Computer de bajo costo lanzado en Reino Unido por la Raspberry Pi Foundation, de toda la familia Raspberry Pi se ha elegido el modelo Zero 2 por ser el de tamaño más reducido pero con una potencia adecuada para el desarrollo de un proyecto como el CanSat. Las características más relevantes para este proyecto son:

- CPU Arm Cortex-A53 de cuatro núcleos y 64 bits a 1 GHz.
- SDRAM de 512 MB.
- LAN inalámbrica de 2,4 GHz 802.11 b/g/n.
- Bluetooth 4.2, Bluetooth Low Energy (BLE), antena integrada.
- Ranura para tarjeta microSD.
- Conector de cámara CSI-2.
- Cabecera GPIO de 40 pines para conexión de periféricos.
- 1 × SPI
- 2 × interfaces I<sup>2</sup>C
- 1 × UART
- Consumo típico de energía: entre 0.7W y 1.5W dependiendo de la carga de trabajo.
- Precio aproximado: 15–20€.

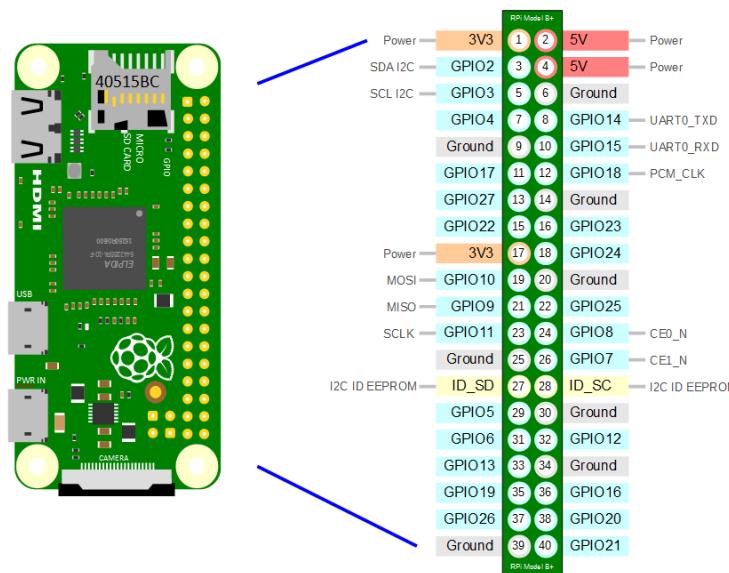


Figura 3.1: Distribución de pines GPIO en Raspberry Pi Zero 2

Como se puede ver en las características la Raspberry Pi Zero 2 cumple con los requisitos necesarios para este proyecto, cuenta con un procesador y memoria adecuados, soporte para cámara (útil para la retransmisión de video en tiempo real), conectividad inalámbrica mediante WiFi y compatibilidad con los protocolos I<sup>2</sup>C y UART mediante los pines GPIO necesarios para la conexión directa de sensores.

- **ESP32:** Es un microcontrolador de bajo coste desarrollado por Espressif Systems que combina bajo coste con buena capacidad de procesamiento y conectividad inalámbrica integrada, lo que lo convierte en una de las opciones más populares para proyectos embebidos, incluyendo CanSat.

A diferencia de un microcomputador como la Raspberry Pi, el ESP32 no ejecuta un sistema operativo generalista, pero su bajo consumo energético y la integración de múltiples periféricos lo hacen convierten en muy buena opción cuando se busca eficiencia y simplicidad del sistema.

Las características más relevantes para este proyecto son:

- CPU dual-core Tensilica Xtensa LX6 a 240 MHz.
- 520 KB de SRAM interna.
- Memoria flash externa: normalmente 4 MB (dependiendo del modelo).
- Conectividad Wi-Fi 802.11 b/g/n.
- Bluetooth 4.2 y BLE.
- 4 × SPI
- 2 × interfaces I<sup>2</sup>C
- 3 × UART
- Hasta 34 pines GPIO (según versión del módulo).
- Consumo típico: entre 0.2 W y 0.6 W, dependiendo del modo de operación.
- Precio aproximado: 4–8€ ESP32.

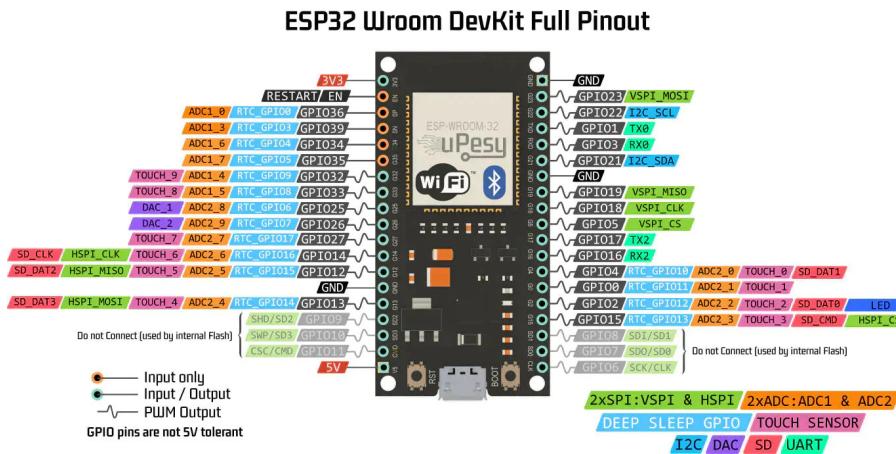


Figura 3.2: Distribución de pines GPIO en ESP32

Gracias a su bajo consumo, potencia y multiples conexiones, el ESP32 permite integrar sensores fácilmente a través de interfaces estándar y puede encargarse tanto de la adquisición como de la transmisión de datos por radio o Wi-Fi. Además, su bajo consumo lo hace especialmente adecuado para sistemas alimentados por batería en entornos con restricciones energéticas. También existen variantes como el ESP32-CAM que integran una cámara de tipo OV2640, lo que permite capturar imágenes y transmitir video mediante Wi-Fi aunque con un rendimiento y resolución inferior a la de Raspberry Pi.

- **Arduino Nano:** Es un microcontrolador compacto de bajo coste basado en el chip ATmega328P, es el más utilizado en entornos educativos gracias a su simplicidad, por lo que cuenta con una amplia comunidad detrás y desarrollo de librerías. A diferencia de la Raspberry Pi Zero 2 o el ESP32, el Arduino Nano no cuenta con conectividad inalámbrica ni capacidad de procesamiento avanzada, pero es suficiente para gestionar sensores básicos y transmitir datos mediante un módulo externo de radio.

Las características más relevantes para este proyecto son:

- Microcontrolador ATmega328P.
- Frecuencia de reloj: 16 MHz.
- Memoria flash: 32 KB (2 KB utilizados por el bootloader).
- SRAM: 2 KB.
- EEPROM: 1 KB.
- 22 pines GPIO (14 digitales, 8 analógicos).
- 1 × interfaz I<sup>2</sup>C.
- 1 × UART.
- 1 × SPI.
- Consumo típico: entre 0.05 W y 0.2 W.
- Precio aproximado: 27€ en la web oficial.

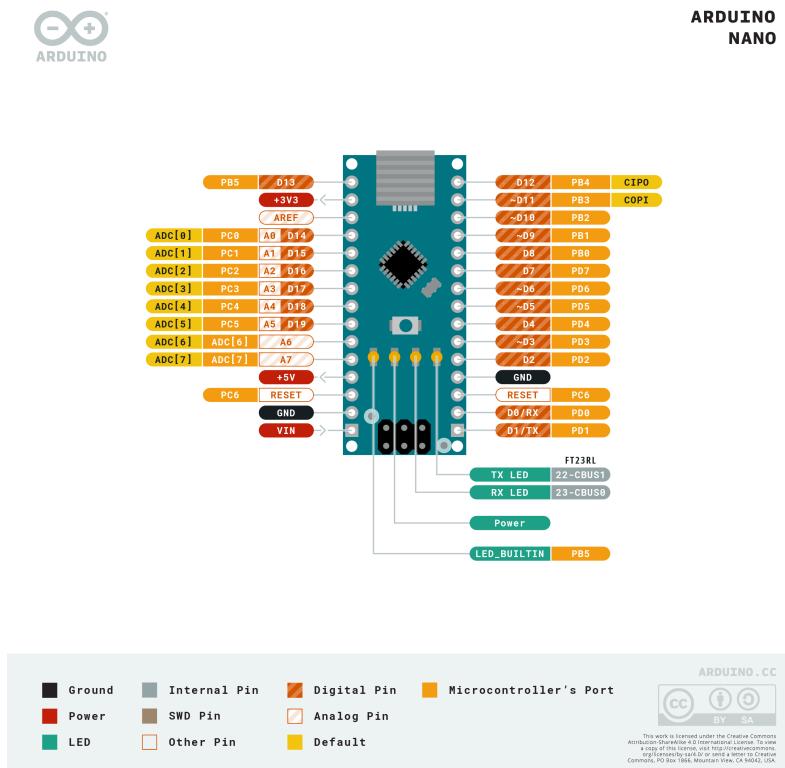


Figura 3.3: Distribución de pines en Arduino Nano

Aunque no tiene las capacidades de procesamiento de una Raspberry Pi ni la conectividad integrada del ESP32, el Arduino Nano puede ser una solución válida para

CanSat muy simples, en los que se priorice el consumo mínimo y no se necesiten funcionalidades avanzadas como WiFi o procesamiento de vídeo, sin embargo, al no tener soporte para cámaras, no cumple con los requisitos técnicos necesarios para este proyecto.

Característica	Raspberry Pi Zero 2	ESP32	Arduino Nano
Procesador	ARM Cortex-A53 (4x, 1 GHz)	Xtensa LX6 (2x, 240 MHz)	ATmega328P (1x, 16 MHz)
Memoria	512 MB SDRAM	520 KB SRAM + 4 MB Flash	2 KB SRAM + 32 KB Flash
Wi-Fi	Sí	Sí	No
Bluetooth	4.2 + BLE	4.2 + BLE	No
SPI	1	4	1
I <sup>2</sup> C	2	2	1
UART	1	3	1
Compatibilidad cámara	CSI (cámara oficial)	OV2640 (ESP32-CAM)	No
Consumo típico	0.7–1.5 W	0.2–0.6 W	0.05–0.2 W
Precio estimado	15–20 €	4–8 €	25–30 €

Tabla 3.1: Comparativa de microcontroladores y microcomputadores

## 3.2. Interfaces de comunicación serie

Una vez analizados sobre los distintos microcontroladores y microprocesadores que existen en el mercado para este tipo de proyectos, es importante entender los distintos tipos de interfaces de comunicación que utilizan para interactuar con sensores y otros módulos externos y como funcionan.

En esta sección se presentan tres de los más relevantes: SPI, I<sup>2</sup>C y UART

- **SPI:** La interfaz SPI (Serial Peripheral Interface) Dhaker (2018) es una interfaz síncrona y full dúplex basada en una arquitectura maestro-esclavo, los dos dispositivos, maestro y esclavo pueden transmitir datos simultáneamente sincronizados con una señal de reloj. La interfaz SPI utiliza cuatro señales:

- Señal de reloj (CLK)
- Selección de chip (CS)
- Salida del maestro hacia el esclavo (MOSI)
- Salida del esclavo hacia el maestro (MISO)

El maestro genera la señal de reloj y controla el intercambio de datos. El pin CS selecciona el estado activo y los pines MISO y MOSI transportan datos en ambas direcciones. Para iniciar la comunicación, el maestro activa el pin CS y empieza a emitir la señal de reloj, al ser una interfaz full dúplex, maestro y esclavo pueden enviar y recibir datos simultáneamente. Esta interfaz tiene un solo maestro y puede tener uno o múltiples esclavos. En configuraciones con múltiples esclavos se pueden conectar de dos modos:

- **Modo regular:** Cada nodo tiene su propia línea de CS
- **Modo cadena (daisy-chain):** Todos los nodos comparten el mismo reloj y CS y los datos se propagan de un esclavo al siguiente. De esta manera se reduce el número de GPIO necesarios en el maestro, aunque aumenta el número de ciclos de reloj requeridos para llegar a cada esclavo.

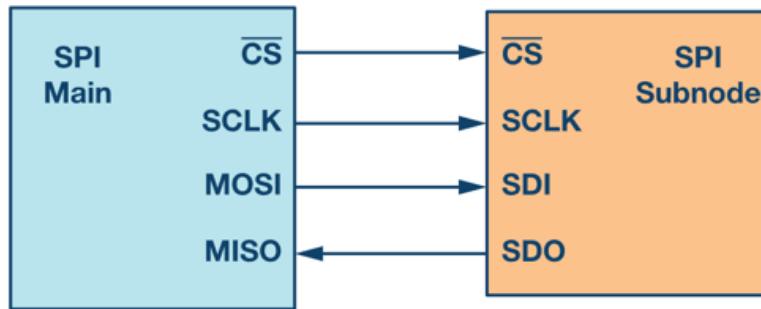


Figura 3.4: Configuración SPI con un maestro y un esclavo

La velocidad de transferencia puede variar dependiendo del hardware utilizado, lo habitual es entre 1 y 10Mbps, aunque algunos dispositivos permiten velocidades superiores.

- **I<sup>2</sup>C:** La interfaz I<sup>2</sup>C (Inter-Integrated Circuit) Semiconductors (2014) es un bus de comunicación síncrono y half dúplex basado en una arquitectura maestro-esclavo, que utiliza solo dos líneas para comunicarse con múltiples dispositivos, originalmente fue desarrollada por Philips Semiconductors en 1982. Esta interfaz utiliza solo dos señales:

- Línea de datos (SDA)
- Línea de reloj (SCL)

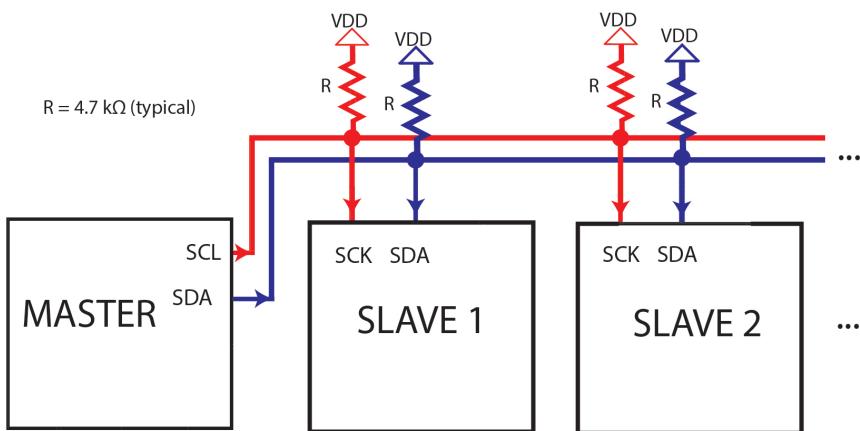


Figura 3.5: Configuración I<sup>2</sup>C con un maestro y varios esclavos

Ambas líneas son bidireccionales aunque al ser half dúplex la comunicación se realiza en un sentido a la vez. Un dispositivo actúa como maestro, iniciando la comunicación y generando la señal de reloj, mientras que uno o más esclavos responden. Cada dispositivo conectado al bus tiene una dirección única.

La comunicación se inicia cuando el maestro envía una condición de inicio, la dirección de uno de los dispositivos conectados al bus y un bit indicando si va a leer o escribir. Despues de transmitir cada byte, el receptor envía una señal de reconocimiento (ACK). La transmisión termina cuando se envía una condición de parada.

El bus I<sup>2</sup>C permite velocidades de transferencia de hasta 100 kbit/s en modo estándar (Standard-mode), 400 kbit/s en modo rápido (Fast-mode), 1 Mbit/s en modo fast-mode plus (Fm+), y hasta 3.4 Mbit/s en modo high-speed (Hs-mode), dependiendo de las capacidades del hardware.

- **UART:** La interfaz UART (Universal Asynchronous Receiver-Transmitter) AG (2018) es una interfaz de comunicación asíncrona basada en una arquitectura punto a punto, que permite la transmisión de datos en serie entre dos dispositivos. A diferencia de las interfaces anteriores, que eran síncronas, UART es asíncrona, por lo que no utiliza una señal de reloj compartida, sino que cada dispositivo funciona con una velocidad de transmisión acordada previamente y común para los dos (baud rate). UART emplea dos líneas de comunicación:

- Transmisión de datos (TX)
- Recepción de datos (RX)

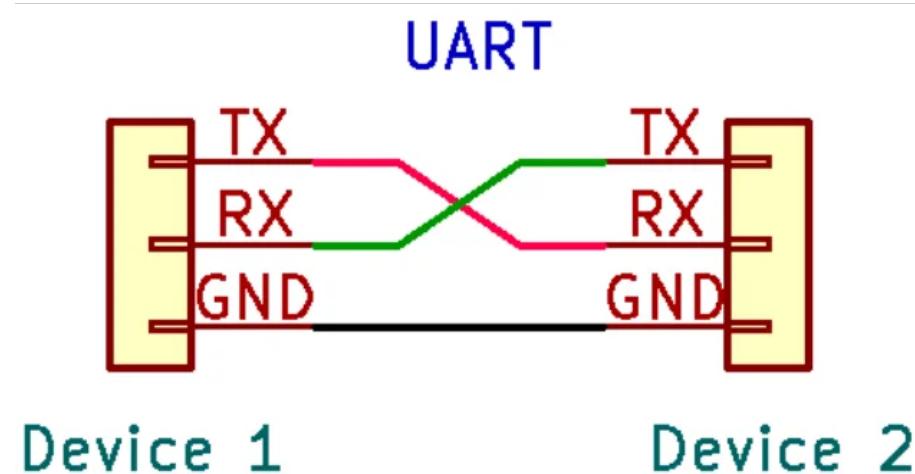


Figura 3.6: Ejemplo de conexión UART

La línea TX de un dispositivo debe ir conectada a la línea RX del otro dispositivo, y la línea RX a TX.

La comunicación es full dúplex, permitiendo la transmisión y recepción de datos de forma simultánea. Cada byte se transmite en una trama que incluye un bit de inicio (start bit), los bits de datos (normalmente 8), un bit opcional de paridad, y uno o más bits de parada (stop bits).

Las velocidades de transmisión habituales van desde 9600 hasta 115200 baudios, aunque se pueden alcanzar velocidades de hasta 1 o 2 Mbps, dependiendo del hardware.

### 3.3. Tecnologías de comunicación por radiofrecuencia en sistemas embebidos

Dentro de los sistemas embebidos y en particular en sistemas tipo CanSat es muy importante la comunicación inalámbrica entre el satélite y la estación de tierra. De entre

Característica	SPI	I <sup>2</sup> C	UART
Tipo de comunicación	Síncrona	Síncrona	Asíncrona
Arquitectura	Maestro-esclavo	Maestro-esclavo	Punto a punto
Número de líneas	4 (CLK, CS, MOSI, MISO)	2 (SDA, SCL)	2 (TX, RX)
Full/Half dúplex	Full dúplex	Half dúplex	Full dúplex
Número de dispositivos	1 maestro, varios esclavos	1 maestro, varios esclavos	Solo dos
Velocidad típica	1–10 Mbps	100 kbit/s – 3.4 Mbit/s	9.6 kbit/s – 3 Mbps
Control de dirección	Señal CS por esclavo	Dirección en el protocolo	No necesario

Tabla 3.2: Comparativa entre las interfaces SPI, I<sup>2</sup>C y UART

las diversas opciones de comunicación inalámbrica, en este apartado vamos a estudiar las distintas opciones que hay en el mercado de comunicación por radiofrecuencia. Este tipo de comunicación nos ofrece un largo alcance manteniendo el bajo consumo, un aspecto muy relevante en un CanSat.

Las tres opciones de comunicación por radiofrecuencia que vamos a estudiar son:

- **LoRa:** La tecnología (Long Range) Augustin et al. (2016) permite una comunicación inalámbrica de largo alcance con un bajo consumo energético, forma parte de las LPWAN (Low-Power Wide-Area Network) redes orientadas a aplicaciones de internet de las Cosas (IoT).

Técnicamente, Lora funciona utilizando una tecnología de modulación de espectro ensanchado basada en chirp spread spectrum (CSS). Cada símbolo (conjunto de bits transmitidos como una única unidad) se transmite mediante un único chirp, es decir, una señal cuya frecuencia varía linealmente (aumentando o disminuyendo) a lo largo del tiempo. El número de bits por símbolo se determina por el Spreading Factor (SF), de modo que un SF de 7 bits codifica 7 bits por símbolo, un SF de 10 codifica 10 bits, y así sucesivamente.

Además, LoRa incorpora un esquema de corrección de errores hacia adelante (Forward Error Correction, FEC), que mejora la fiabilidad de la comunicación en entornos con ruido o interferencias. Este sistema añade bits redundantes a los datos transmitidos, permitiendo al receptor detectar y corregir errores sin necesidad de retransmisión. La cantidad de redundancia depende de la tasa de codificación (Coding Rate, CR), que puede configurarse entre 4/5 y 4/8. Esta tasa indica cuántos bits útiles se transmiten por cada bloque de bits totales. Por ejemplo, una tasa de 4/5 significa que por cada 5 bits enviados, 4 contienen información útil y 1 es redundante para corrección de errores. De forma similar, una tasa de 4/8 implica que solo 4 de cada 8 bits son datos útiles y los 4 restantes son bits de corrección. Cuanto menor es la tasa, mayor es la redundancia y, por tanto, mayor la robustez frente a errores, aunque la velocidad efectiva de transmisión es menor.

Opera en bandas de frecuencia no licenciadas, como 433 MHz, 868 MHz (Europa) o 915 MHz (América), esto permite su uso sin coste dentro de este espectro.

El alcance varía dependiendo del entorno, desde cientos de metros en entornos urbanos con muchas interferencias hasta más de 15 kilómetros en entornos abiertos y despejados.

- **XBee:** Los módulos XBee International han sido desarrollados por Digi International y permiten establecer comunicaciones inalámbricas de corto a medio alcance con bajo consumo energético. Utilizan el estándar IEEE 802.15.4 para las capas física y MAC,

lo que les proporciona interoperabilidad con otros dispositivos compatibles con este estándar.

Estos módulos operan en la banda ISM de 2.4GHz utilizando modulación DSSS (Direct Sequence Spread Spectrum) con O-QPSK (Offset Quadrature Phase Shift Keying), alcanzando una tasa de transferencia aérea de 250kbps.

La técnica *DSSS* consiste en dispersar cada bit de datos sobre una secuencia de mayor ancho de banda mediante una secuencia pseudoaleatoria (chipping code), lo que proporciona robustez frente a interferencias y permite que múltiples transmisiones compartan el mismo canal sin colisiones significativas.

Por su parte, *O-QPSK* es una variante de la modulación por desplazamiento de fase en cuadratura (QPSK), que introduce un desfase temporal entre las componentes en fase y en cuadratura para reducir los cambios bruscos en la señal transmitida, mejorando así la eficiencia espectral y reduciendo la probabilidad de errores durante la demodulación.

Digi ha desarrollado interfaces de configuración y modos de operación propios, como el modo API, que encapsula datos y comandos en tramas estructuradas para control avanzado, además del modo transparente, que permite una comunicación directa punto a punto.

XBee permite topologías de red como punto a punto o estrella. Cada módulo tiene una dirección MAC única de 64 bits y puede asignarse una dirección corta de 16 bits para identificación dentro de la red. La comunicación con el microcontrolador suele realizarse mediante UART, con velocidades configurables de hasta 1Mbps.

En condiciones óptimas, el alcance puede variar entre 30 y 100 metros en interiores, y superar los 300 metros en exteriores.

- **APC220:** El módulo APC220 Technologies (2010) es un transceptor de radiofrecuencia desarrollado por Appcon Technologies que permite establecer comunicaciones inalámbricas punto a punto en sistemas embebidos de bajo consumo. Este módulo opera en la banda ISM de 433 MHz lo que hace que no necesite licencia para funcionar.

El APC220 emplea modulación GFSK (Gaussian Frequency Shift Keying), una variante de FSK (Frequency Shift Keying) en la que las transiciones de frecuencia entre los niveles binarios (0 y 1) se suavizan usando un filtro gaussiano aplicado previamente a la señal digital. Esta técnica reduce la anchura de banda de la señal transmitida y minimiza las emisiones fuera de banda, lo que se traduce en menor interferencia con otros sistemas y una mayor eficiencia espectral. Además, GFSK mejora la robustez frente al ruido, mejorando su eficiencia en entornos con interferencias.

La tasa de transmisión es configurable entre 1200 bps y 19200 bps, utilizando UART como interfaz con el microcontrolador.

El módulo integra un amplificador de potencia y un receptor de alta sensibilidad, permitiendo un alcance de hasta 1000 metros en condiciones óptimas con línea de visión clara. Además, incluye un circuito interno de corrección de errores y control automático de ganancia (AGC), que mejoran la fiabilidad de la comunicación.

Su configuración puede ajustarse mediante comandos AT o mediante una herramienta software proporcionada por el fabricante, conectando el módulo a través de un convertidor USB-UART

Característica	LoRa	XBee (802.15.4)	APC220
Frecuencia de operación	433/868/915 MHz	2.4 GHz	433 MHz
Modulación	CSS (Chirp Spread Spectrum)	DSSS + O-QPSK	GFSK
Velocidad de datos	0.3–27 kbps	250 kbps	1.2–19.2 kbps
Alcance típico	Hasta 15 km (entornos abiertos)	30–300 m	Hasta 1 km
Corrección de errores	FEC configurable (CR 4/5–4/8)	No especificado	FEC + AGC internos
Interfaz con MCU	UART	UART	UART
Topología de red	Punto a punto	Punto a punto / estrella	Punto a punto

Tabla 3.3: Comparativa de tecnologías de comunicación por radiofrecuencia

### 3.4. Sensores empleados para telemetría: presión barométrica, IMU y GNSS

A continuación se describen de los distintos tipos sensores empleados para cumplir con los requisitos del CanSat. Estos sensores son los encargados de tomar mediciones precisas sobre distintas variables del entorno. Para cumplir con estos requisitos se requiere la integración de tres tipos distintos de sensores:

- **Presión barométrica:** Los sensores de presión barométrica se utilizan para medir la presión atmosférica y de esta forma estimar la altura sobre el nivel del mar mediante modelos estándar como el ISA (International Standard Atmosphere) Skybrary (2021).

La presión atmosférica es la fuerza que ejerce el peso de una columna de aire sobre un área determinada, por ello, al medir la presión sobre un punto de la tierra con mayor altitud la presión será menor por ser menor la cantidad de aire sobre ese punto.

Distintos factores meteorológicos pueden hacer variar la presión atmosférica, principalmente con los cambios de temperatura, al variar la temperatura, varía la densidad del aire y, por lo tanto, varía su peso, afectando a la presión.

Otros factores como la humedad relativa o el viento también influyen aunque de menor manera y pueden ser obviados.

En el mercado se pueden encontrar distintos modelos de estos sensores, que varían entre ellos en función de su precisión, interfaz de conexión con el microcontrolador, consumo energético y precio.

Sensor	Rango de presión	Precisión	Interfaz	Consumo típico	Precio (€)
BMP388	300–1250 hPa	±8 Pa (±0.66 m)	I <sup>2</sup> C, SPI	3.4 µA	~3.50
BME280	300–1100 hPa	±12 Pa (±1 m)	I <sup>2</sup> C, SPI	2.7 µA	~4.00
MPL3115A2	50–1100 hPa	±0.04 hPa (±0.3 m)	I <sup>2</sup> C	40 µA	~6.00

Tabla 3.4: Comparativa de sensores de presión barométrica

- **IMU:** Una unidad de medición inercial o IMU, es un dispositivo que mide la velocidad, orientación y fuerzas gravitacionales de un aparato, para hacerlo generalmente utiliza una combinación de sensores, acelerómetros y giroscopios.

Para su funcionamiento utiliza tres acelerómetros colocados de tal forma que sus ejes de medición queden de manera ortogonal entre sí, estos acelerómetros se utilizan para medir las fuerzas de aceleración (fuerzas G) que actúan en cada eje.

También cuenta con tres giroscopios colocados en un patrón ortogonal similar, estos giroscopios miden la velocidad angular alrededor de cada eje, permitiendo conocer

los cambios en la orientación del objeto respecto a un sistema de referencia.

Algunos modelos también pueden contar con un magnetómetro, que mide el campo magnético terrestre y permite estimar el rumbo respecto al norte magnético.

La combinación de estos sensores nos permite conocer la orientación en el espacio de un objeto en términos de los tres ángulos de Euler: pitch, yaw, y roll muy utilizados en aplicaciones aeronáuticas y espaciales.

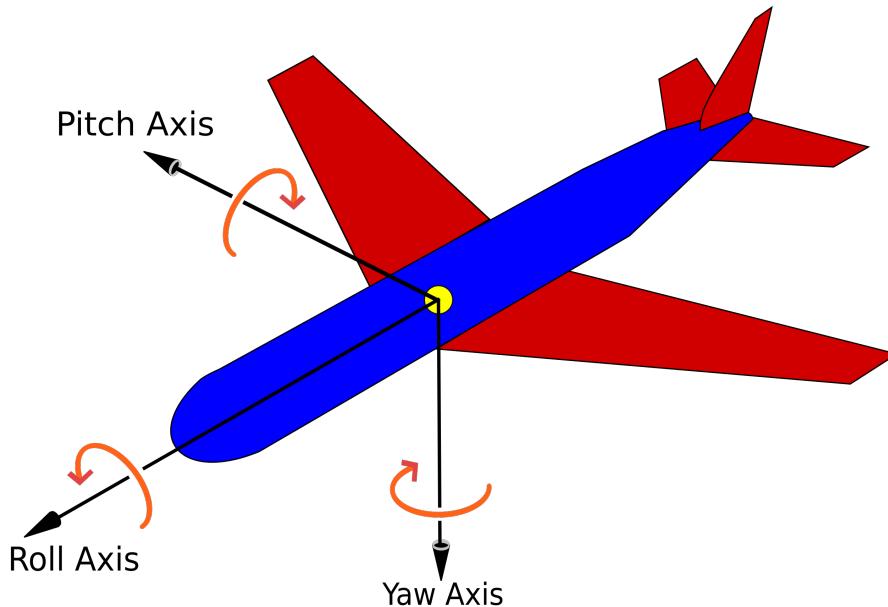


Figura 3.7: Sistema de referencia basado en los ángulos de pitch, yaw y roll

En algunos modelos de sensores disponibles en el mercado, esta combinación de sensores viene acompañada de un procesador interno que realiza fusión sensorial, permitiendo obtener directamente la orientación del dispositivo sin necesidad de cálculos externos.

Sensor	Componentes	Salida de orientación	Interfaz	Consumo típico	Precio (€)
MPU6050	Acelerómetro + Giroscopio	No (requiere procesamiento externo)	I <sup>2</sup> C	3.9 mA	~1.50
BNO055	Acelerómetro + Giroscopio + Magnetómetro	Sí (procesamiento interno con fusión)	I <sup>2</sup> C / UART	12 mA	~9.00
BNO085	Acelerómetro + Giroscopio + Magnetómetro	Sí (mejor precisión que BNO055)	I <sup>2</sup> C / UART / SPI	3.5 mA	~14.00
LSM9DS1	Acelerómetro + Giroscopio + Magnetómetro	No (requiere fusión externa)	I <sup>2</sup> C / SPI	1.0 mA	~5.00

Tabla 3.5: Comparativa de sensores IMU

- **GNSS:** Global Navigation Satellite System Labs (2024) es un sistema de navegación por satélite que engloba las siguientes constelaciones de satélites: GPS (Estados Unidos) U.S. Government (2024), Galileo (Europa) for the Space Programme (2024), GLONASS (Rusia) Federation (2024) y BeiDou (China) Office (2024).

Estos sistemas se utilizan para determinar la posición geográfica y la velocidad de un objeto en cualquier parte del planeta utilizando estas constelaciones de satélites. Para determinar la posición y velocidad se necesita un receptor GNSS que recibe las señales emitidas por los satélites que contienen información sobre su posición y la fecha exacta en la que fueron enviadas. Cuando el receptor recibe señales de

Sistema	País / Región	Satélites operativos	Precisión típica (civil)	Frecuencias principales
GPS	Estados Unidos	31	5–10 metros	L1, L2, L5
Galileo	Unión Europea	28	<1 metro (con E5 AltBOC)	E1, E5a, E5b, E6
GLONASS	Rusia	24	5–10 metros	L1, L2
BeiDou	China	45	2.5–5 metros	B1, B2, B3

Tabla 3.6: Comparativa de las principales constelaciones GNSS

al menos tres satélites, puede calcular su posición usando una técnica geométrica llamada trilateración García (2006).

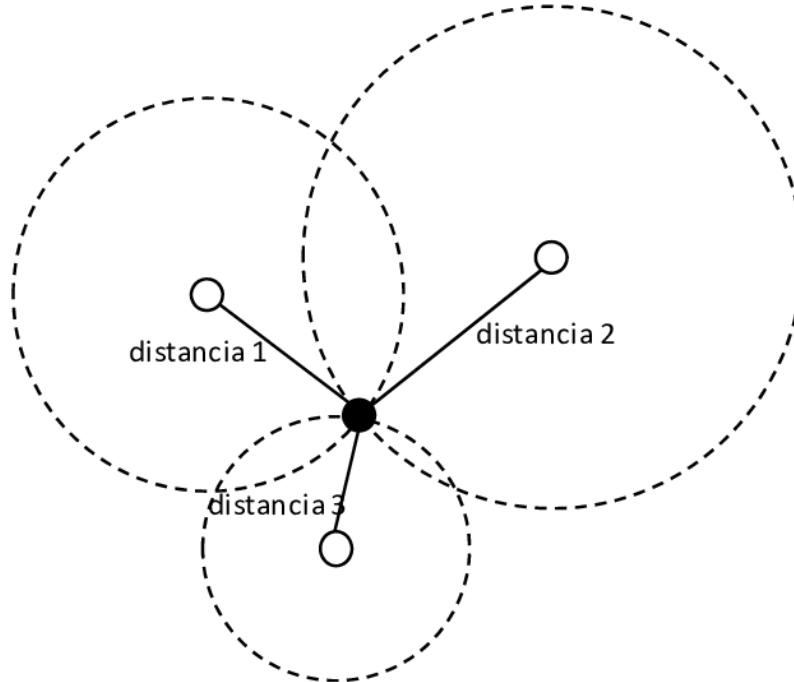


Figura 3.8: Ejemplo de cálculo de la posición usando la técnica trilateración

Esta técnica permite conocer la posición de un objeto conociendo su distancia a tres puntos de referencia, en este caso, estos tres puntos son las posiciones de los satélites.

Actualmente, en el mercado existe una gran variedad de receptores GNSS diseñados para aplicaciones embebidas, que varían en su precisión, consumo, constelaciones soportadas y coste. A continuación se muestra una comparativa entre algunos de los receptores GNSS más comunes.

Módulo	Constelaciones soportadas	Precisión típica	Frecuencia de actualización	Consumo	Precio aprox.
BN-880	GPS, GLONASS, Galileo, BeiDou	2m CEP	1–10Hz	50mA a 5V	15–25€
NEO-M8N	GPS, GLONASS, Galileo, BeiDou	2m CEP	hasta 18Hz	<150mA a 5V	35–40€
NEO-F9P	GPS, GLONASS, Galileo, BeiDou (con RTK)	cm-level con RTK	hasta 20Hz	100mA a 3.3V	110–130€

Tabla 3.7: Comparativa de receptores GNSS comunes en sistemas embebidos

### 3.5. Captura y transmisión de vídeo en tiempo real

Como parte de los requisitos para este proyecto CanSat se encuentra la transmisión de vídeo, pudiendo ser visualizada en tiempo real desde una plataforma web.

Para llevar a cabo esto, estudiaremos los fundamentos técnicos de la captura digital de video, las distintas maneras de códificarla, los distintos formatos de contenedor y los principales protocolos de transmisión en tiempo real.

### 3.5.1. Captura digital de vídeo

La captura digital de vídeo es el proceso mediante el cual una cámara convierte la luz del entorno en información digital que puede ser procesada y almacenada. Este proceso se realiza mediante un sensor de imagen, que transforma la luz en señales eléctricas.

Existen principalmente dos tipos de sensores usados en este tipo de sistemas de este tipo:

- **CMOS:** Los sensores CMOS (Complementary Metal-Oxide-Semiconductor) (Nahalingam y Katehi, 2023), fueron desarrollados en 1993 como una alternativa más eficiente y económica a los sensores CCD

Funcionan mediante una matriz de fotodioides que convierten la luz incidente en carga eléctrica. Cada píxel del sensor incorpora circuitos activos que amplifican y digitalizan la señal generada, lo que permite leer simultáneamente múltiples píxeles en paralelo.

Los sensores CMOS actuales cuentan con conversores analógico-digitales (ADC) y procesado de imagen incluidos en el propio chip, lo que disminuye el ruido y mejora el rendimiento en condiciones de baja iluminación.

- **CCD:** Los sensores CCD (Charge-Coupled Device) (Blanc y Deltel, 2001) fueron desarrollados en 1969 por Willard Boyle y George E. Smith.

A diferencia de los sensores CMOS, en los CCD la luz incide sobre una matriz de fotodioides que convierte los fotones en carga eléctrica, pero esta carga no se convierte inmediatamente en una señal digital. En su lugar, se transfiere secuencialmente a lo largo del chip mediante registros de desplazamiento hasta llegar a una única etapa de conversión analógico-digital

Esto lo convierte en un proceso más lento y con mayor consumo energético.

Dentro de la captura de video encontramos algunos parámetros importantes:

- **Resolución:** número de píxeles de cada fotograma (por ejemplo, 640x480 o 1920x1080) y determina el nivel de detalle de la imagen.
- **Tasa de fotogramas (fps):** indica el número de fotogramas por segundo, y afecta a la fluidez del vídeo.

Una vez obtenidos los datos del sensor, se sigue un proceso de codificación del vídeo, en este proceso se comprime los datos brutos para reducir su tamaño pero manteniendo una buena calidad de imagen.

Estos son los codecs (formatos de compresión) más comunes:

- **H.264 (AVC):** El códec H.264 (también conocido como AVC, Advanced Video Coding) Wiegand et al. (2003) es un estándar de compresión de vídeo con pérdida,

Divide cada fotograma en macrobloques, que se analizan para aprovechar redundancias temporales (entre fotogramas) y espaciales (dentro de un mismo fotograma).

Usando esta técnica, puede alcanzar una alta relación de compresión sin comprometer demasiado la calidad de imagen.

- **H.265 (HEVC):** El códec H.265 o HEVC (High Efficiency Video Coding) Sze et al. (2014) es el sucesor directo de H.264, está diseñado para ofrecer el mismo nivel de calidad que su antecesor, utilizando aproximadamente la mitad del bitrate.

Para ello utiliza bloques de compresión más grandes y mayor profundidad de predicción, además soporta resoluciones de hasta 8K, para lograr esto necesita mayor capacidad de procesamiento, tanto al codificar como al decodificar.

- **MJPEG (Motion JPEG):** mjp (2021), Codifica cada fotograma de vídeo como una imagen JPEG independiente.

A diferencia de los códecs anteriores no se beneficia de las redundancias entre fotogramas, sin embargo, es más simple y rápido, por lo que necesita menos capacidad de procesamiento y provoca una menor latencia.

Códec	Compresión	Requiere procesamiento elevado	Latencia
H.264 (AVC)	Alta	Media	Baja
H.265 (HEVC)	Muy alta	Alta	Media
MJPEG	Baja	Baja	Muy baja

Tabla 3.8: Comparativa entre códecs de vídeo comunes

Los codecs definen como se comprime la imagen, pero no como se almacenan esos datos, para ello se utilizan formatos contenedores.

Un formato contenedor es un formato de archivo que encapsula uno o más flujos de datos codificados, normalmente video y audio, junto con la información necesaria para sincronizarlos y reproducirlos correctamente.

Algunos de los formatos contenedores más comunes incluyen:

- **AVI (Audio Video Interleave):** Formato contenedor desarrollado por Microsoft en 1992. Presenta limitaciones para la transmisión en tiempo real y no soporta bien múltiples pistas de audio o subtítulos.
- **MP4 (MPEG-4 Part 14):** Uno de los formatos más utilizados actualmente. Soporta video (principalmente H.264/H.265), audio, subtítulos y metadatos. Es ideal para transmisión y almacenamiento, y es compatible con casi todos los navegadores y plataformas.
- **MKV (Matroska):** Formato contenedor libre y flexible, capaz de contener múltiples flujos de video, audio y subtítulos en un solo archivo. Es muy utilizado para contenidos de alta calidad, aunque su compatibilidad es más limitada.
- **FLV (Flash Video):** Fue diseñado originalmente para la transmisión de video a través de Adobe Flash. Soporta codecs como H.264 y es ampliamente utilizado en plataformas streaming. Se utiliza sobre todo en sistemas que emplean el protocolo RTMP debido a su baja latencia.

Formato	Códecs comunes	Soporte multitrack	Uso típico
AVI	H.264, MJPEG, DivX	Limitado	Almacenamiento local
MP4	H.264, H.265 (HEVC), AAC	Sí	Transmisión y almacenamiento
MKV	H.264, H.265, VP9, Opus	Sí	Contenido multimedia de alta calidad
FLV	H.264, MP3, AAC	Limitado	Transmisión en tiempo real (RTMP)

Tabla 3.9: Comparativa de formatos contenedores de vídeo

### 3.5.2. Protocolos de transmisión de vídeo en tiempo real

Los protocolos en tiempo real permiten enviar vídeo y audio a través de redes IP de manera continua, permitiendo la visualización del contenido con la menor latencia posible.

A continuación se describen los principales protocolos utilizados en este contexto:

- **RTMP (Real-Time Messaging Protocol):** El protocolo RTMP (Macromedia, 2012) funciona sobre TCP, normalmente usando el puerto 1935 y establece una conexión persistente entre cliente y servidor.

Utiliza multiplexación a nivel de protocolo para dividir el flujo continuo en pequeños (chunks) de datos que intercalan vídeo, audio y comandos de control, lo que permite mantener una única conexión TCP. Este diseño permite mantener una latencia baja, normalmente entre 3 y 5 segundos.

La conexión comienza con un intercambio de mensajes (handshake) entre cliente y servidor, seguido por comandos codificados en formato AMF para controlar la transmisión.

A diferencia de protocolos como HLS o DASH que segmentan el vídeo en archivos, RTMP divide el flujo en chunks internos multiplexados con vídeo, audio y control.

- **HLS (HTTP Live Streaming):** El protocolo HLS (Inc., 2009–2017 (IETF draft) fue desarrollado por Apple para transmitir vídeo sobre HTTP.

Divide el contenido en pequeños fragmentos de vídeo (por defecto de unos 6 segundos), que se almacenan como archivos independientes y se listan en un archivo de índice (playlist) en formato M3U8. Los clientes descargan y reproducen estos fragmentos secuencialmente, permitiendo el streaming adaptativo según el ancho de banda disponible.

Al utilizar HTTP, HLS es altamente compatible con servidores web estándar, aunque introduce mayor latencia (típicamente entre 6 y 30 segundos) que otros protocolos como RTMP.

- **MPEG-DASH (Dynamic Adaptive Streaming over HTTP):** MPEG-DASH (ISO/IEC, 2022) es un estándar internacional (ISO/IEC 23009-1:2012) para streaming adaptativo sobre HTTP.

Funciona fragmentando el contenido de vídeo o audio en segmentos pequeños junto con un archivo de manifiesto (MPD) que describe las diferentes versiones de bitrate disponibles. El cliente descarga estos fragmentos secuencialmente y ajusta dinámicamente la calidad según el ancho de banda disponible.

Como funciona sobre HTTP, funciona bien con todos los navegadores web. Aunque introduce más latencia que RTMP, permite ajustar la calidad del vídeo en función la conexión, lo que mejora el rendimiento en redes con ancho de banda variable.

A diferencia de HLS, no está ligado a una implementación concreta como la de Apple.

- **WebRTC (Web Real-Time Communication):** WebRTC (W3C y IETF, 2021) es un conjunto de tecnologías desarrolladas por el W3C y el IETF que permite la comunicación en tiempo real directamente entre navegadores.

Funciona sobre UDP y está diseñado para ofrecer baja latencia en aplicaciones como videollamadas, transmisión en directo o compartición de pantalla. Utiliza protocolos como SRTP para cifrar el contenido y ICE/STUN/TURN para gestionar la conexión punto a punto incluso a través de NAT o firewalls.

A diferencia de otros protocolos como HLS o DASH, WebRTC no usa HTTP ni archivos fragmentados, sino que transmite los datos como un flujo continuo mediante canales de comunicación directos entre pares (peer-to-peer). Esto le permite alcanzar latencias muy bajas, típicamente inferiores a 500ms, pero es más complejo de usar en sistemas embebidos.

Protocolo	Transporte	Latencia típica	Segmentación
RTMP	TCP (puerto 1935)	3–5 s	Chunks multiplexados
HLS	HTTP (TCP)	5–15 s	Archivos segmentados (.ts, .m3u8)
DASH	HTTP (TCP)	4–10 s	Archivos segmentados (MP4, .mpd)
WebRTC	UDP	<500 ms	Flujo continuo peer-to-peer

Tabla 3.10: Comparativa de protocolos de transmisión de vídeo en tiempo real

### 3.6. Visualización de datos en tiempo real: arquitecturas orientadas a eventos

Para permitir la visualización en tiempo real de los datos de telemetría del CanSat desde una interfaz web, es necesario diseñar una arquitectura orientada a eventos que gestione el flujo de datos de forma asíncrona y con baja latencia.

#### 3.6.1. Definición de arquitecturas orientadas a eventos

Una arquitectura orientada a eventos es un sistema en el que los distintos componentes se comunican entre sí mediante mensajes transmitidos de manera asíncrona.

En lugar de realizar peticiones constantes para consultar el estado del sistema (como ocurre en arquitecturas basadas en polling), los consumidores de eventos esperan a que el sistema les notifique cuando hay nuevos datos disponibles. Esto permite reducir la latencia y el consumo de recursos, sobre todo en aplicaciones que requieren actualizaciones en tiempo real.

Frente a las arquitecturas monolíticas, donde todos los componentes están fuertemente acoplados, las arquitecturas orientadas a eventos fomentan un diseño desacoplado, en el que los emisores y receptores de eventos pueden evolucionar de manera independiente.

#### 3.6.2. Flujo típico de eventos hacia el cliente web

En una arquitectura orientada a eventos, es habitual que los datos generados por dispositivos o servicios se publiquen primero en un sistema de mensajería como una cola o un bus de eventos.

El flujo habitual de eventos suele incluir los siguientes pasos:

1. El evento es generado por un productor (por ejemplo, un sensor o sistema embebido) y publicado en un sistema de mensajería (como RabbitMQ, MQTT o Kafka).
2. Un servidor backend actúa como consumidor del sistema de mensajería, recibiendo los eventos conforme se generan.
3. El backend reenvía estos eventos al cliente web en tiempo real utilizando un protocolo de comunicación asíncrona entre cliente-servidor como Websockets.

Este enfoque desacopla la generación de eventos del consumo en el navegador y permite una visualización en tiempo real con baja latencia.

### 3.6.3. RabbitMQ

RabbitMQ es un sistema de mensajería basado en colas que implementa el protocolo AMQP (Advanced Message Queuing Protocol) Pivotal Software (2023). Su función principal es desacoplar la comunicación entre productores y consumidores de datos, permitiendo que los mensajes se almacenen temporalmente en una cola hasta que el consumidor esté listo para procesarlos.

En este modelo, los productores publican mensajes en una exchange, que se encarga de enrutar esos mensajes a una o varias colas dependiendo del tipo de intercambio (direct, topic, fanout...). Los consumidores, por su parte, se suscriben a una o varias colas para recibir los mensajes.

RabbitMQ soporta distintos tipos de exchange, que determinan cómo se enrutan los mensajes a las colas:

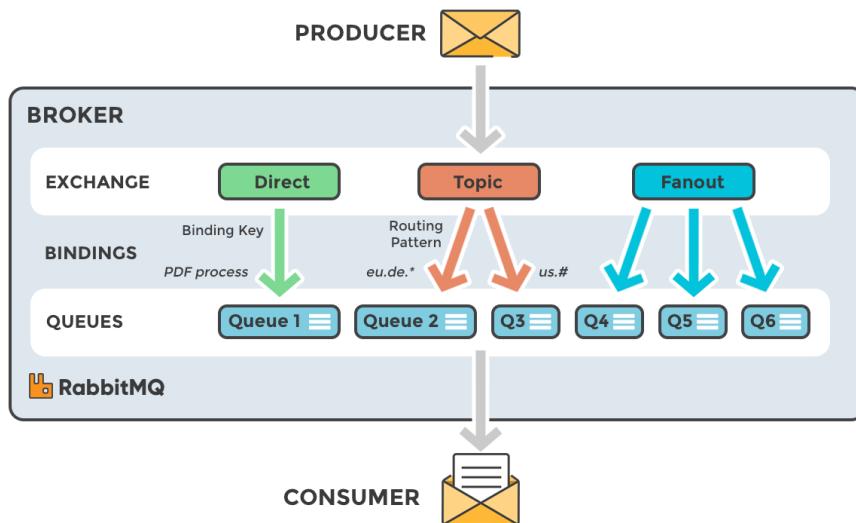


Figura 3.9: Ejemplo de los distintos tipos de exchange de RabbitMQ

- **Direct:** envía el mensaje solo a las colas que tienen una clave de enrutamiento (routing key) que coincide exactamente con la del mensaje.

- **Topic:** permite un enrutamiento más flexible usando patrones con comodines en la clave de enrutamiento. Útil para suscripciones parciales (ej., sensors.temperature.\*).
- **Fanout:** reenvía el mensaje a todas las colas enlazadas, sin considerar la clave de enrutamiento. Ideal para difusiones o broadcasts.
- **Headers:** usa los campos del encabezado del mensaje en lugar de la clave de enrutamiento para decidir el destino.

Este sistema garantiza la entrega ordenada, confiable y asíncrona de los datos, siendo especialmente útil en arquitecturas orientadas a eventos donde los distintos componentes pueden fallar o estar temporalmente desconectados.

### 3.6.4. WebSockets

WebSocket es un protocolo de comunicación definido por el estándar RFC 6455 (Fette y Melnikov, 2011) que permite establecer una conexión persistente y bidireccional entre un cliente (normalmente un navegador) y un servidor.

A diferencia del modelo HTTP tradicional basado en peticiones-respuestas, WebSocket permite que ambos extremos envíen datos en cualquier momento, lo que lo hace ideal para aplicaciones en tiempo real. Una vez establecida la conexión, los datos se transmiten en forma de frames ligeros.

Gracias a esta arquitectura full-duplex, WebSocket es ampliamente utilizado en aplicaciones que requieran actualización inmediata sin recurrir a técnicas como polling.

# Capítulo 4

## Diseño e implementación del sistema

Este capítulo describe la arquitectura general del sistema y detalla cada uno de sus componentes, desde el montaje electrónico del CanSat hasta la visualización de los datos en tiempo real.

Se justifica la elección del hardware y software empleados, se explica el funcionamiento del código embebido, y se documenta la integración entre los distintos módulos del sistema: la Raspberry Pi, el backend en Spring Boot, la base de datos, el broker de eventos RabbitMQ y el frontend desarrollado en Flutter.

Finalmente, se presentan las pruebas realizadas para verificar el funcionamiento del sistema completo.

### 4.1. Selección de componentes y tecnologías utilizadas

Este apartado describe los principales elementos hardware y software empleados en el sistema.

La selección se ha basado en los requisitos técnicos del proyecto y en las conclusiones del análisis comparativo desarrollado en el capítulo anterior.

- **Unidad de procesamiento:** se ha utilizado una Raspberry Pi Zero 2 W.



Figura 4.1: Raspberry Pi Zero 2 W

Su capacidad para ejecutar Linux, capturar y codificar vídeo por hardware, y gestionar interfaces como I<sup>2</sup>C, UART y CSI, la hacen adecuada para su integración directa en un CanSat sin necesidad de microcontroladores adicionales.

- **Sensor de presión barométrica:** el BMP388 Adafruit (2023) permite estimar la altitud mediante medición barométrica. Además de la presión atmosférica, proporciona la temperatura del entorno. Ofrece una precisión de  $\pm 8$  Pa, bajo consumo y comunicación por I<sup>2</sup>C.

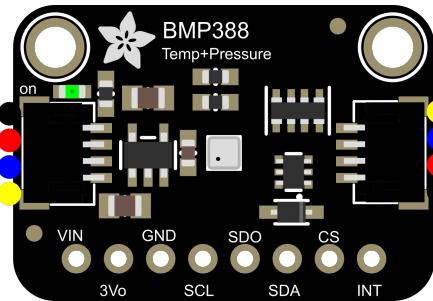


Figura 4.2: Sensor de presión barométrica bmp388

- **IMU:** Se ha empleado el BNO085 Adafruit (2021), un sensor de 9 grados de libertad que combina acelerómetro, giroscopio y magnetómetro. Incorpora un procesador dedicado que realiza la fusión sensorial internamente y entrega directamente la orientación absoluta, lo que evita cálculos adicionales en la Raspberry Pi.

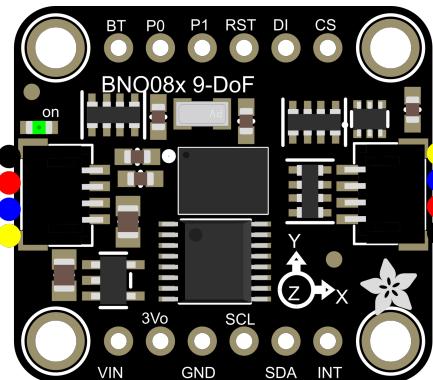


Figura 4.3: IMU bno085

- **GNSS:** se ha utilizado el módulo BN-880 bn8 (2023), que integra un receptor GNSS con soporte para múltiples constelaciones (GPS, GLONASS, Galileo y BeiDou). Proporciona datos de posición, velocidad y altitud en tiempo real mediante interfaz UART. Incluye una antena activa integrada y una brújula electrónica.



Figura 4.4: GPS bn-880

- **Cámara:** se utiliza el modelo oficial de la Raspberry Pi Raspberry Pi Camera Module v2 (2016) con interfaz CSI. Permite la captura de vídeo codificado en H.264 mediante la GPU, sin comprometer el rendimiento del sistema.



Figura 4.5: Raspberry Pi Camera Module 2

- **Transmisión de datos:** el sistema utiliza Wi-Fi si hay red disponible, en caso contrario el módulo LoRa E32-900T20D EBYTE (2024). Este módulo opera en 868 MHz y permite comunicación a larga distancia mediante UART.



Figura 4.6: Módulo LoRa E32-900T20D

- **Antena LoRa:** se utiliza una antena externa de 868–915 MHz, con conector SMA macho, ganancia de 3 dBi e impedancia de  $50 \Omega$ .



Figura 4.7: Antena LoRa U.FL a SMA 868–915 MHz, 3 dBi

- **Alimentación:** el sistema se alimenta mediante una batería recargable de ion de litio (Li-ion) tipo 18650 de 3,7 V conectada a un cargador MCP73871 Microchip Technology (2019), que permite la recarga mediante panel solar y proporciona protección frente a sobrecarga, descarga profunda y cortocircuitos.

Para obtener los 5 V requeridos por la Raspberry Pi, se emplea un convertidor elevador DC-DC ajustable (modelo VISSQH), que incrementa la tensión de entrada hasta una salida estabilizada.

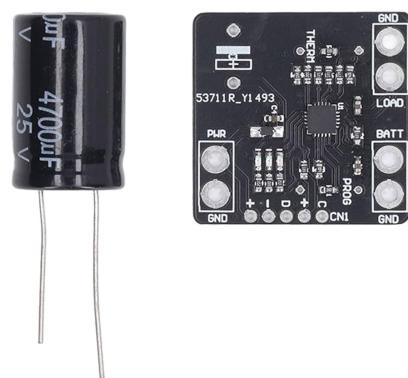


Figura 4.8: Cargador MCP73871



Figura 4.9: Batería 18650

- **Estación de tierra:** está formada por un receptor LoRa conectado mediante UART a una Raspberry Pi 4.

Esta actúa como pasarela entre el CanSat y el sistema de mensajería RabbitMQ, reenviando los datos recibidos a través de LoRa.

- **Software embebido:** tanto la Raspberry Pi usada en el CanSat como la de la estación de tierra ejecutan scripts desarrollados en Python.
- **Software del sistema:** el backend está desarrollado en Java utilizando el framework Spring Boot, que facilita la construcción de servicios web REST y la integración con sistemas de mensajería.

Para el intercambio de mensajes se emplea RabbitMQ, un sistema de colas que implementa el protocolo AMQP.

La persistencia de datos se gestiona con PostgreSQL, un sistema de gestión de bases de datos relacional.

La interfaz de usuario está implementada en Flutter, un framework multiplataforma desarrollado en Dart, y establece la comunicación en tiempo real mediante WebSocket utilizando el protocolo STOMP.

## 4.2. Arquitectura general del sistema

La arquitectura del sistema se organiza en tres bloques principales: el CanSat, la estación de tierra y la infraestructura de backend y visualización web.

Esta arquitectura sigue un enfoque orientado a eventos, en el que los datos generados por el CanSat son publicados y transmitidos hacia los consumidores mediante mecanismos de mensajería asincrónica.

Los tres componentes principales son:

- **CanSat:** integra sensores, módulo GNSS, cámara y comunicaciones.

Una Raspberry Pi Zero 2 W adquiere y empaqueta los datos de telemetría para su envío. Cuando hay red Wi-Fi disponible, los eventos se publican directamente en un broker RabbitMQ. En caso contrario, los datos se transmiten mediante LoRa a la estación de tierra.

Todos los datos de telemetría adquiridos por los sensores se guardan de manera local en formato JSON. En cuanto a la retransmisión de vídeo en tiempo real, solo se realiza cuando está conectado a una red Wi-Fi, en caso contrario, el vídeo se guarda en la memoria de la Raspberry Pi.

- **Estación de tierra:** compuesta por una Raspberry Pi 4 conectada al receptor LoRa mediante UART.

Ejecuta un script en Python que interpreta los paquetes recibidos y los reenvía al broker RabbitMQ a través de la red.

Esta estación actúa como pasarela cuando no hay conectividad directa entre el CanSat y la infraestructura de backend en los casos en los que no existe conexión directa mediante red.

- **Infraestructura de visualización:** esta infraestructura está formada por varios componentes:
  - **RabbitMQ:** recibe los eventos de telemetría desde la estación de tierra o directamente desde el CanSat y los transmite al backend.
  - **Backend:** se encarga de consumir los eventos con los datos de telemetría que llegan desde RabbitMQ, guardarlos en la base de datos para su posterior descarga, y emitir los eventos hasta el frontend utilizando websockets.
  - **Base de datos:** se trata de una base de datos relacional PostgreSQL, en ella se guardan todos los eventos de telemetría asociados a su fecha de recepción.
  - **Frontend:** es una aplicación web desarrollada en Flutter, cuenta con distintos tipos de gráficas para visualizar los datos del CanSat en tiempo real a través de WebSockets.
  - **Servidor de vídeo en tiempo real:** recibe el flujo de vídeo codificado desde el CanSat mediante el protocolo RTMP y lo adapta para su reproducción en navegadores web. La transmisión solo se realiza cuando hay conectividad Wi-Fi disponible.

Para mejorar la portabilidad y facilitar el despliegue de la infraestructura en distintos entornos, todos los servicios del backend, la base de datos, el broker de mensajería y el servidor de vídeo se han contenerizado utilizando Docker.

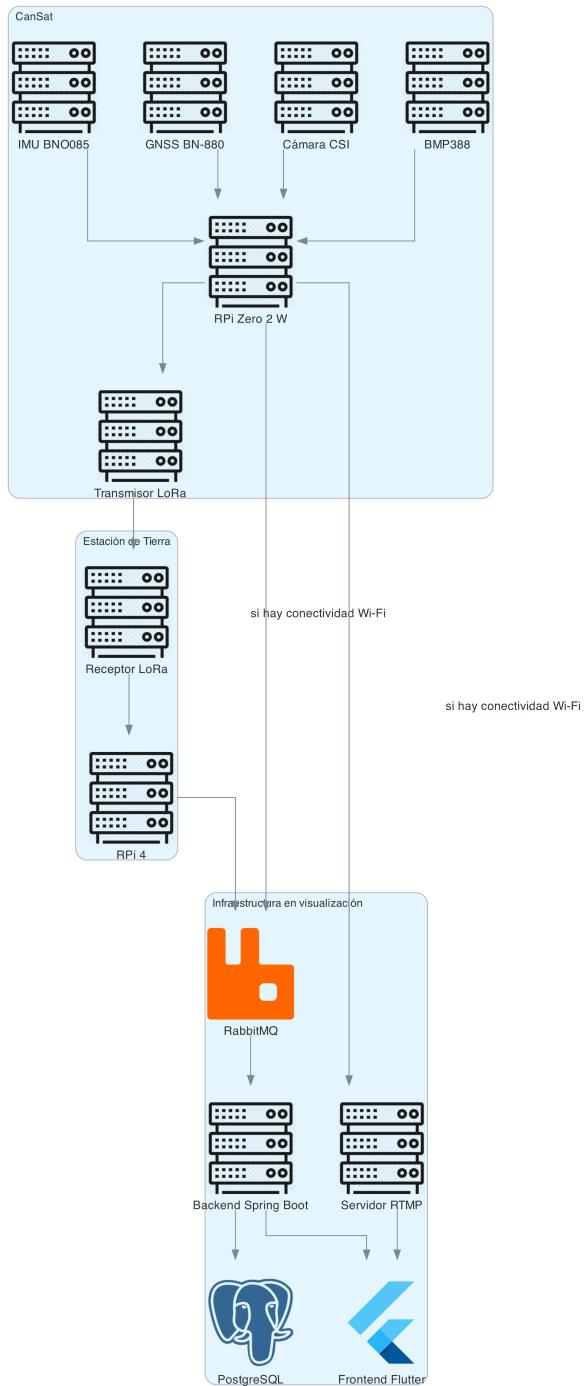


Figura 4.10: Arquitectura general del sistema

### 4.3. Montaje electrónico del CanSat

El montaje electrónico se realizó inicialmente sobre una protoboard, con el objetivo de validar las conexiones y el funcionamiento de los distintos módulos durante la fase de desarrollo.

Cada componente fue integrado mediante cableado directo a la Raspberry Pi Zero 2 W, usando las interfaces específicas de cada uno (UART, I<sup>2</sup>C).

A continuación, se describen las conexiones realizadas entre los distintos periféricos y la Raspberry Pi, incluyendo los pines utilizados y las interfaces empleadas.

- **Sensor de presión BMP388:** utiliza la interfaz I<sup>2</sup>C.
  - SDA: GPIO 2 (pin físico 3)
  - SCL: GPIO 3 (pin físico 5)
  - Alimentación: 5 V (pin físico 2)
  - GND: pin físico 6
- **IMU BNO085:** también conectada por I<sup>2</sup>C en el mismo bus que el BMP388.
  - SDA: GPIO 2 (pin físico 3)
  - SCL: GPIO 3 (pin físico 5)
  - Alimentación: 5 V (pin físico 2)
  - GND: pin físico 6
- **Módulo LoRa E32-900T20D:** comunica por UART.
  - TX del módulo → RX de la Pi: GPIO 15 (pin físico 10)
  - RX del módulo ← TX de la Pi: GPIO 14 (pin físico 8)
  - Alimentación: 5 V (pin físico 2)
  - GND: pin físico 6
  - Pines M0 y M1: conectados respectivamente a los GPIO 23 (pin físico 16) y GPIO 24 (pin físico 18) de la Raspberry Pi, lo que permite gestionar el modo de operación del módulo desde software.
  - Conexión de antena: la antena externa se conecta al conector SMA hembra del módulo LoRa, permitiendo la comunicación en la banda de 868–915 MHz.

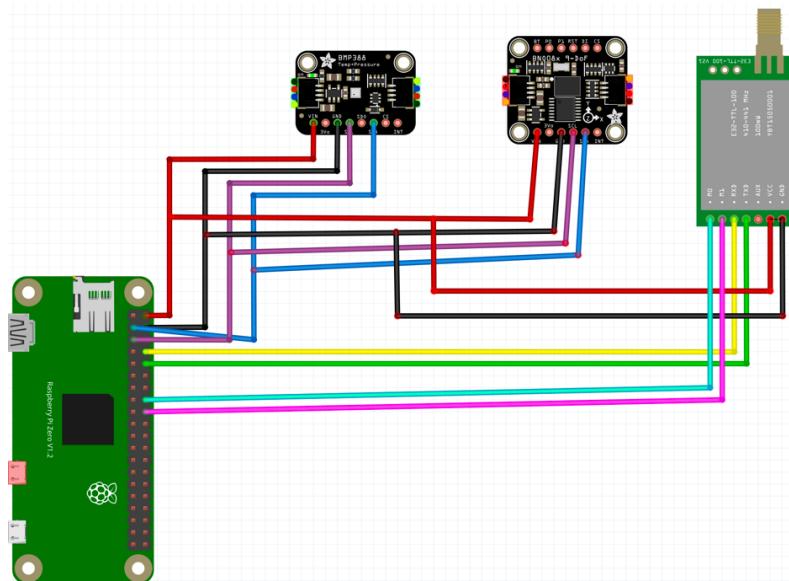


Figura 4.11: Diagrama de conexión de los sensores y módulo LoRa

- **GNSS BN-880:** comunica por UART. Dado que la Raspberry Pi Zero 2 W solo dispone de un puerto UART hardware accesible por GPIO (usado en este caso por el módulo LoRa), se ha optado por conectar el BN-880 a través de un adaptador USB a UART.

El módulo utilizado es un conversor basado en el chip CP2102, que se conecta al puerto micro-USB de la Raspberry Pi y proporciona una interfaz UART adicional accesible desde el sistema operativo como un puerto /dev/ttyUSB0.

Dado el espacio reducido disponible, se ha modificado el adaptador CP2102 retirando el conector USB macho original y sustituyéndolo por un conector micro-USB macho, soldado directamente a la placa. Esto permite conectarlo al puerto micro-USB de datos de la Raspberry Pi de forma más compacta, sin necesidad de adaptadores adicionales.

- TX del GNSS → RX del adaptador
- RX del GNSS ← TX del adaptador
- Alimentación: 5 V conectado al pin VCC del adaptador
- GND: pin físico 14 conectado al GND del adaptador
- Conexión de datos: vía micro-USB del adaptador al puerto USB de la Raspberry Pi

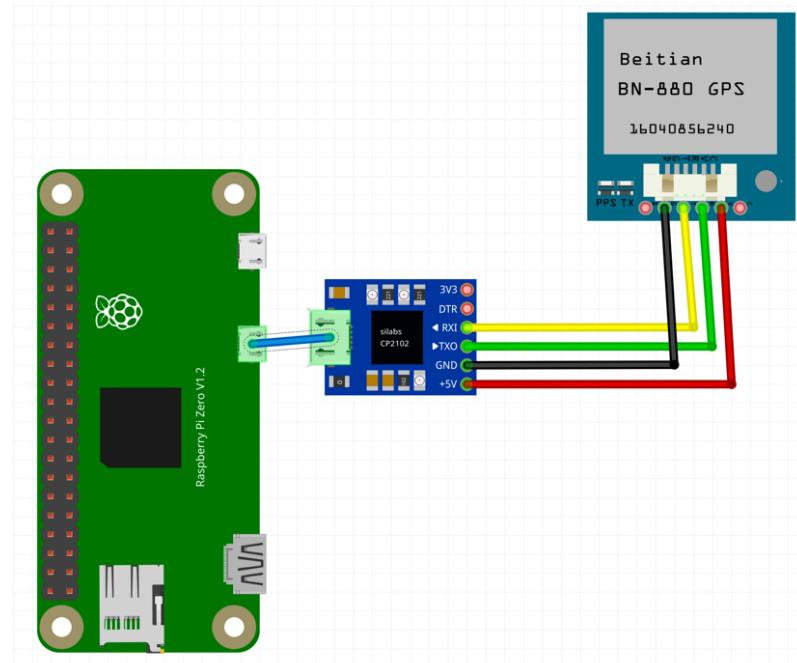


Figura 4.12: Diagrama de conexión del módulo GNSS BN-880 mediante el adaptador CP2102

- **Cargador MCP73871:** es el encargado de gestionar la alimentación del sistema y la recarga de la batería.

Este módulo permite cargar una batería de ion de litio y, simultáneamente, alimentar el sistema, conmutando automáticamente entre la entrada de alimentación externa y la batería según su disponibilidad.

- **Entrada de carga:** al conector USB-C hembra del módulo se conecta una entrada de alimentación externa.

En este caso, se utilizan tres paneles solares de 2 V conectados en serie, cuyo cableado termina en un conector USB-C macho compatible con la entrada del cargador.

- **Conexión de la batería:** una batería de ion de litio de 3,7 V (tipo 18650) se conecta directamente a los terminales BAT+ y BAT- del módulo.

Esta conexión permite tanto la carga de la batería como el suministro de energía al sistema cuando no hay entrada externa disponible.

- **Salida VBUS:** el terminal VBUS proporciona la tensión de salida activa en cada momento (procedente de la entrada USB-C o de la batería).

Esta salida no está regulada y puede variar entre 3,7 V y 5,5 V, por lo que se conecta a un convertidor elevador (boost converter) que estabiliza la tensión a 5 V.

La salida del boost se conecta al pin físico 2 (5 V) de la Raspberry Pi, y su masa al pin físico 39 (GND), proporcionando alimentación estable a todo el sistema.

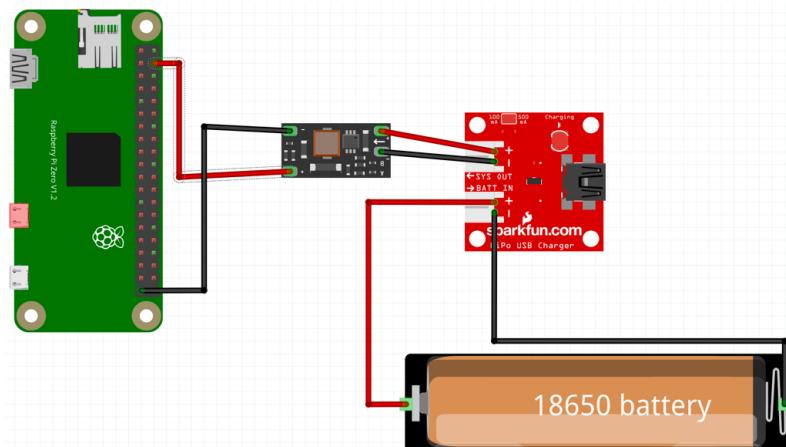


Figura 4.13: Diagrama de conexión del cargador MCP73871 y booster

- **Cámara CSI:** conectada directamente al puerto CSI (Camera Serial Interface) de la Raspberry Pi Zero 2 W mediante un cable plano.

El puerto CSI es una interfaz específica para cámaras que permite transmitir vídeo sin utilizar pines GPIO. Además, proporciona tanto la alimentación como la señal de datos a través del propio conector.

Módulo	Función	GPIO	Pin físico
BMP388	SDA	GPIO 2	3
	SCL	GPIO 3	5
	VCC	-	1 (3.3 V)
	GND	-	6
BNO085	SDA	GPIO 2	3
	SCL	GPIO 3	5
	VCC	-	1 (3.3 V)
	GND	-	6
LoRa E32-900T20D	TX	GPIO 15	10
	RX	GPIO 14	8
	M0	GPIO 23	16
	M1	GPIO 24	18
	VCC	-	1 (5 V)
	GND	-	6
GNSS BN-880	TX → RX del CP2102	-	-
	RX ← TX del CP2102	-	-
	VCC → VCC del CP2102	-	-
	GND → GND del CP2102	-	-
Adaptador CP2102	UART USB	-	Puerto USB (micro)
MCP73871 (cargador)	Entrada (USB-C)	-	-
	Salida a booster	-	-
	Boost out → Pi	-	2 (5 V), 39 (GND)
Cámara CSI	Datos + alimentación	-	CSI

Tabla 4.1: Resumen de conexiones de los periféricos a la Raspberry Pi Zero 2 W

A continuación se describe también el montaje de la estación de tierra, formada por un módulo LoRa conectado a una Raspberry Pi 4 a través de la interfaz UART:

- TX del módulo → RX de la Pi: GPIO 15 (pin físico 10)
- RX del módulo ← TX de la Pi: GPIO 14 (pin físico 8)
- Pines M0 y M1: conectados respectivamente a los GPIO 23 (pin físico 16) y GPIO 24 (pin físico 18) de la Raspberry Pi, lo que permite gestionar el modo de operación del módulo desde software.
- Alimentación: 5 V (pin físico 2)
- GND: pin físico 6
- Conexión de antena: la antena externa se conecta al conector SMA hembra del módulo LoRa, permitiendo la comunicación en la banda de 868–915 MHz.

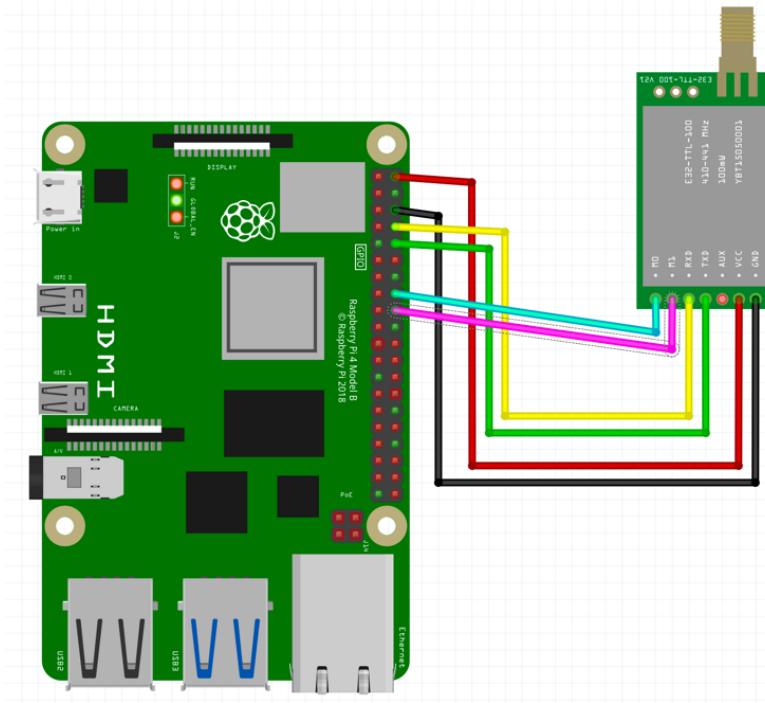


Figura 4.14: Diagrama de conexión de la estación de tierra con Raspberry Pi 4 y módulo LoRa

Elemento	Función	GPIO	Pin físico
LoRa E32-900T20D	TX → RX de la Pi	GPIO 15	10
	RX ← TX de la Pi	GPIO 14	8
	M0	GPIO 23	16
	M1	GPIO 24	18
	VCC	—	2 (5 V)
	GND	—	6

Tabla 4.2: Conexiones del módulo LoRa a la Raspberry Pi 4 en la estación de tierra

Tras verificar el correcto funcionamiento de todas las conexiones en la protoboard, se procedió a su traslado a una placa PCB, diseñada específicamente teniendo en cuenta las dimensiones y restricciones del CanSat.

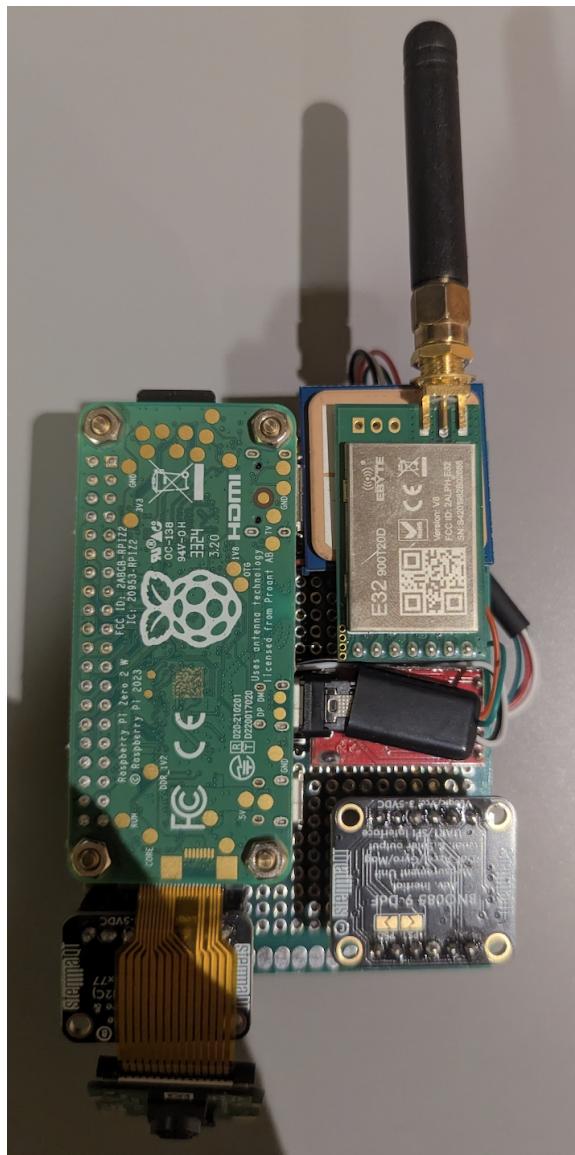


Figura 4.15: Montaje final del sistema en la PCB (vista frontal)

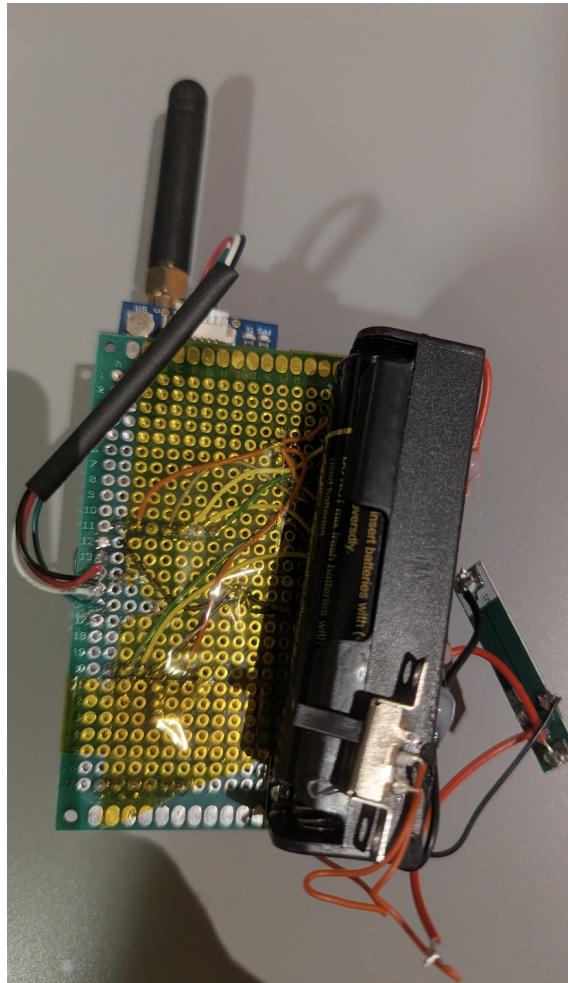


Figura 4.16: Montaje final del sistema en la PCB (vista trasera)

#### 4.4. Código embebido en la Raspberry Pi

El código embebido desarrollado para la Raspberry Pi Zero 2 W está implementado en Python y se encarga de gestionar de forma concurrente la captura de vídeo, la adquisición de datos de sensores, el envío de eventos de telemetría por LoRa, y la publicación online a través de RabbitMQ.

El flujo de inicialización del CanSat sigue los siguientes pasos:

- Esperar una conexión a internet durante 30 segundos para saber si debe enviar los datos mediante LoRa o publicarlos directamente en la cola de RabbitMQ.
- Crear una carpeta local donde se almacenarán los datos y vídeos generados, con nombre basado en la fecha y hora de arranque.
- Iniciar los módulos de captura de vídeo y envío de telemetría en hilos independientes.

#### 4.4.1. Captura de vídeo en tiempo real

La clase **Camera** gestiona la captura de vídeo a través de la interfaz CSI usando la librería Picamera2.

El vídeo se guarda localmente y, si hay internet disponible, también se retransmite en directo mediante ffmpeg a un servidor RTMP externo.

El flujo de vídeo se configura con una resolución de 640x480 a 15 FPS, y se codifica en tiempo real con h264\_v4l2m2m para su emisión.

#### 4.4.2. Adquisición y envío de telemetría

La clase **TelemetrySender** recopila información de múltiples sensores conectados a la Raspberry Pi:

- **BMP388:** mide presión y temperatura atmosférica. Se accede a través de la librería adafruit\_bmp3xx.
- **BNO085:** proporciona la orientación del dispositivo (pitch, roll, yaw) a partir de cuaterniones. Se utiliza la librería adafruit\_bno08x.
- **BN-880 GNSS:** ofrece la posición geográfica, altitud GPS, velocidad y número de satélites. La información se decodifica a través de las librerías pyserial y pynmea2.

Estos datos se agregan en un evento del tipo TM con la fecha en la que se genera, este evento realiza el siguiente flujo:

- Se generan cada 200 ms, agregando los datos más recientes de los sensores.
- Se publican en una cola RabbitMQ si hay conexión a internet.
- Se envían por LoRa mediante la clase **LoraSender** si no hay conexión a internet (máximo un evento cada 2 segundos).
- Se almacenan localmente en un fichero .jsonl.

#### 4.4.3. Envío por LoRa

La clase **LoraSender** se encarga del envío de eventos en modo texto (CSV) a través del módulo LoRa. Este envío se realiza en un hilo dedicado para no bloquear la adquisición de datos.

Después de enviar un evento sistema espera una confirmación OK durante un máximo de 2 segundos para considerar el envío exitoso y enviar el siguiente, si no ese evento se descarta y se procede a enviar el siguiente, para ello se hace uso de una cola local.

#### 4.4.4. Conexión con RabbitMQ

La clase **RabbitmqConnectionManager** crea y mantiene la conexión con el servidor RabbitMQ.

Publica los eventos codificados en JSON en el exchange tracking\_device\_events de tipo fanout. Si la conexión se pierde, intenta restablecerla automáticamente.

#### 4.4.5. Formato de los eventos

Cada evento tiene los siguientes campos principales:

- **type:** tipo de evento (actualmente sólo TM para telemetría).
- **datetime:** fecha de la generación del evento formato ISO.
- **payload:** contiene todos los datos de sensores disponibles en ese instante.

Los eventos se serializan en formato JSON o CSV, en función del canal de transmisión utilizado (RabbitMQ o LoRa, respectivamente)

### 4.5. Software de la estación de tierra

El software de la estación de tierra está desarrollado en python y se encarga de realizar las siguientes acciones:

- Intenta detectar si hay conexión a internet durante 30 segundos. Si se detecta conexión, el código continúa, en caso contrario, finaliza con un mensaje de error.
- Establece la conexión con el broker de RabbitMQ utilizando la clase `RabbitmqConnectionManager`.
- Espera hasta 30 segundos a que el puerto serie `/dev/serial0` esté disponible y sea accesible.

Una vez disponible, abre el puerto a una velocidad de 9600 baudios, este es el puerto en el que recibe los mensajes del módulo LoRa.

- Lee continuamente desde el puerto serie, acumulando los datos en un búfer hasta que detecta el delimitador `$`, que indica el final de un mensaje.
- Intenta parsear el mensaje como un evento válido en formato CSV, usando la clase `Event`.

Si el parseo se produce sin errores, convierte el evento a JSON y lo publica en RabbitMQ y responde por LoRa con un OK para confirmar la recepción.

- En caso de que el mensaje recibido está mal formado, lo descarta y registra el error sin interrumpir la ejecución.
- El sistema se puede detener mediante una interrupción por teclado (ctrl+C). Al cerrarse, libera el puerto serie y cierra la conexión con RabbitMQ de forma segura.

### 4.6. Backend con Spring Boot

El backend del sistema está desarrollado en Java utilizando el framework Spring Boot, es el encargado de recibir los eventos mediante RabbitMq, guardarlos en PostgreSQL y reenviarlos mediante WebSockets.

#### 4.6.1. Recepción de eventos

Los eventos generados por el CanSat son enviados a RabbitMQ utilizando un exchange de tipo fanout llamado `tracking_device_events`. El backend está suscrito a este exchange a través de una queue denominada `tracking_device_events.dashboard_backend`, tal y como se configura en la clase `RabbitmqConfiguration`:

Cada vez que se recibe un mensaje en esta cola, el componente `TrackingDeviceEventsConsumer` se encarga de:

1. Deserializar el evento desde JSON a un objeto `Event`.
2. Guardar el evento en la base de datos a través del repositorio `EventJPA`.
3. Publicar el evento a través de WebSocket mediante la clase `TrackingDeviceEventsProducer`.

#### 4.6.2. Persistencia de eventos

Los eventos se almacenan en una base de datos relacional mediante Spring Data JPA. Cada evento se representa con la clase `Event`, la cual contiene:

- Un identificador único (`id`).
- El tipo de evento (`type`).
- La fecha y hora del evento (`datetime`).
- Un payload genérico como mapa clave-valor (`Map<String, String>`), almacenado como JSON en la base de datos.

#### 4.6.3. Acceso a eventos

Se proporciona un endpoint REST para consultar eventos entre dos fechas:

- `/events/range`: devuelve una lista de eventos como JSON.
- `/events/range/download`: permite descargar los eventos en formato JSONL.

#### 4.6.4. Distribución en tiempo real

El backend incluye soporte para WebSocket, los eventos almacenados se reenvían al canal `/topic/events` a través de la clase `TrackingDeviceEventsProducer`.

### 4.7. Frontend con Flutter para visualización en tiempo real

El frontend ha sido desarrollado utilizando Flutter, permitiendo desplegar una aplicación web responsive capaz de visualizar los eventos emitidos por el CanSat. La comunicación con el backend se realiza a través de WebSocket, suscribiéndose al canal `/topic/events`.

La interfaz ha sido diseñada con el objetivo de ser lo más genérica posible, facilitando la visualización de métricas y gráficas sin atarse a una configuración de sensores específica o a un diseño de CanSat concreto. Esto permite adaptar fácilmente la plataforma a otros proyectos con diferentes sensores o estructuras.

#### 4.7.1. Arquitectura general

La aplicación se estructura en torno a un patrón BLoC (Business Logic Component), donde el estado del último evento recibido se propaga de forma reactiva a los distintos widgets encargados de representar la telemetría.

El componente principal `TrackingDeviceEventBloc` inicializa el cliente de WebSocket, gestiona la conexión y parsea los mensajes entrantes desde el backend, convirtiéndolos en instancias de la clase `Event`. Esta clase encapsula tanto la fecha como el tipo de evento y su carga útil (payload).

#### 4.7.2. Visualización modular

La interfaz gráfica se divide en varios componentes:

- **Metrics:** muestra el último valor de todos los parámetros y una lista de todos los eventos de telemetría recibidos.

También permite la descarga de eventos históricos en formato .jsonl especificando un rango de fechas.



Figura 4.17: Ejemplo del Componente Metrics

- **Attitude:** renderiza un modelo tridimensional que rota en tiempo real según los valores de yaw, pitch y roll incluidos en cada evento, de esta manera es posible ver la orientación actual del CanSat.



Figura 4.18: Ejemplo del Componente Attitude

- **TelemetryChart:** representa todos los valores de los parámetros recibidos como gráficas temporales utilizando, permitiendo observar la evolución de múltiples variables simultáneamente.

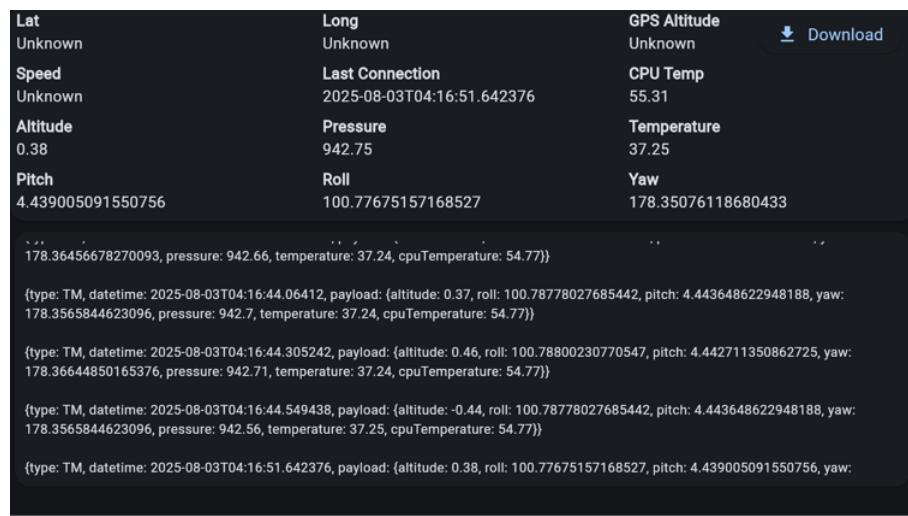


Figura 4.19: Ejemplo del Componente TelemetryChart

- **Map:** muestra un mapa con la última ubicación recibida del CanSat.

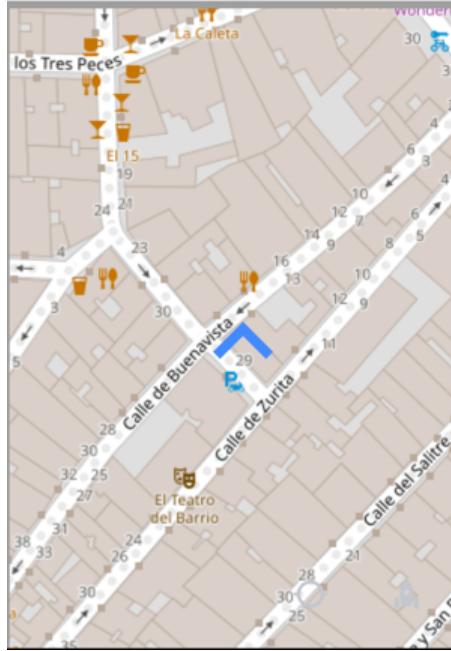


Figura 4.20: Ejemplo del Componente Map

- **Camera:** muestra la retransmisión de video en directo.

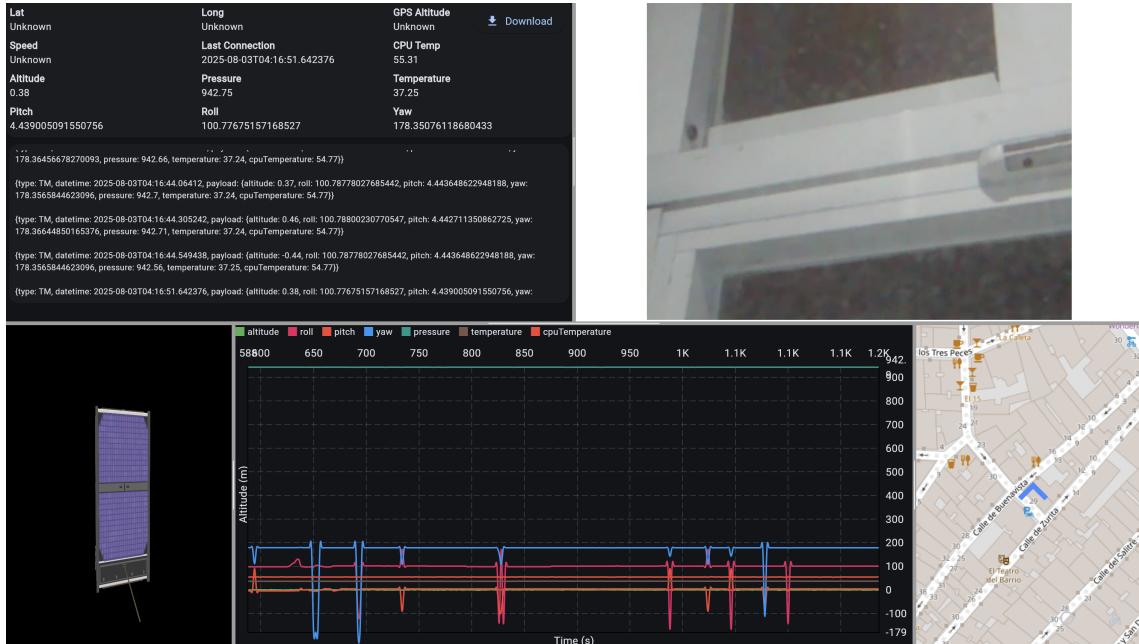


Figura 4.21: Interfaz en modo escritorio

#### 4.7.3. Adaptabilidad

El diseño de la interfaz se adapta automáticamente a la resolución de pantalla. En escritorio el espacio se divide en secciones horizontales y verticales que pueden ser redimensionadas de manera independiente, mientras que en móviles los elementos se muestran

uno debajo de otro.

## 4.8. Pruebas de integración y validación

Después de la realización de todo el desarrollo del sistema, se realizaron pruebas para comprobar su correcto funcionamiento, desde la adquisición de datos en el CanSat hasta su visualización en el aplicación web.

Durante estas pruebas se validó:

- La generación y envío de eventos de telemetría por LoRa y RabbitMQ según la disponibilidad de internet.
- El almacenamiento de los eventos en la base de datos del backend y su retransmisión por WebSocket.
- La correcta actualización en tiempo real de las métricas, gráficas y visualizaciones en el frontend.
- La duración de la batería en condiciones reales, confirmando que el sistema puede operar de forma autónoma durante aproximadamente 4 horas.
- El alcance de la antena LoRa, validando la recepción de datos a más de 500 metros en campo abierto.



# Capítulo 5

## Conclusiones y Trabajo Futuro

El objetivo principal que se planteaba al inicio del trabajo de crear una plataforma de visualización de datos en tiempo real, reutilizable y desacoplada de hardware específico ha sido cumplido. Esto se ha conseguido gracias al análisis de otras soluciones existentes y al diseño de una arquitectura centrada, desde sus primeras fases, en la independencia del hardware y en su integración sencilla en otros proyectos.

La arquitectura modular del sistema facilita su mantenimiento y evolución. Cada componente (captura de datos, transmisión, backend, frontend) puede modificarse de manera independiente sin afectar al resto del sistema, lo que permite su reutilización parcial.

La construcción de un CanSat ha servido como caso ejemplo para validar la plataforma, confirmando que no existen dependencias entre el sistema de visualización y el hardware. Este ejemplo demuestra que la solución puede adaptarse fácilmente a configuraciones de sensores, protocolos o arquitecturas de CanSat distintas.

Además, el desarrollo del flujo completo, desde la adquisición de datos hasta su visualización en tiempo real, ha permitido identificar puntos críticos, permitiendo optimizar los tiempos de transmisión y garantizar la sincronización de eventos en entornos reales.

En conjunto, el trabajo ha generado una plataforma flexible, útil no solo como solución cerrada, sino como punto de partida para futuros desarrollos en proyectos con necesidades similares de visualización de datos.

A pesar de haber alcanzado los objetivos inicialmente propuestos, se han identificado varias líneas de mejora en las que trabajar en el futuro:

- Aumentar el número de gráficas disponibles en la aplicación web, incorporando visualizaciones específicas para determinados parámetros.
- Implementar un sistema de envío de telecomandos al CanSat para manejar su configuración de manera remota, esto permitiría activar o desactivar sensores o modificar parámetros como el intervalo de envío de datos directamente en tiempo de ejecución.
- Implementar un sistema de autenticación en la aplicación web para proteger posibles datos sensibles.
- Añadir soporte para visualizar múltiples CanSat al mismo tiempo. Actualmente no se distingue el origen de los eventos, por lo que no es posible monitorizar varios dispositivos simultáneamente.

- El desarrollo del CanSat se ha centrado en la parte electrónica y de software, se podría crear una carcasa 3D y un paracaídas para poder usarlo en lanzamientos reales.

Chapter 6

## Introduction

Introduction to the subject area. This chapter contains the translation of Chapter 1.



Chapter **7**

## Conclusions and Future Work

Conclusions and future lines of work. This chapter contains the translation of Chapter 5.



# Bibliografía

- Mjpeg (motion jpeg) video codec. Library of Congress Preservation Formats, 2021.
- Beitian bn-880 gnss compass module. <https://ardupilot.org/copter/docs/common-beitian-gps.html>, 2023.
- ADAFRUIT. *Adafruit 9-DOF Orientation IMU Fusion Breakout – BNO085*, 2021.
- ADAFRUIT. Adafruit bmp388 - precision barometric pressure and altimeter sensor. <https://www.adafruit.com/product/3966>, 2023.
- AG, I. T. *Component UART V2.50 - Software Module Datasheet*, 2018.
- AMERICAN ASTRONAUTICAL SOCIETY. American astronautical society – advancing space science and exploration. Electronic resource, sitio oficial, 2025.
- AMERICAN CANSAT COMPETITION. Cansat competition — design-build-fly aerospace challenges. Electronic resource, official website, 2025.
- ARDUINO NANO. Arduino nano. <https://store.arduino.cc/products/arduino-nano>, ????
- AUGUSTIN, A., YI, J., CLAUSEN, T. y TOWNSLEY, W. M. A study of lora: Long range & low power networks for the internet of things. *Sensors*, vol. 16(9), páginas 1466–1518, 2016.
- BLANC, N. y DELTEL, C. Review of ccd technologies for digital photography. *Photogrammetric Week*, 2001. Accessed July 2025.
- CORPORATION, S. Lora: Long range, low power wireless platform. *White Paper*, 2015.
- DHAKER, P. Introduction to spi interface. *Analog Dialogue*, vol. 52, 2018.
- EBYTE. E32-900t20d sx1276 lora wireless module (868mhz, 20dbm uart). <https://www.cdebyte.com/products/E32-900T20D/1>, 2024.
- ESP32. Esp32 series datasheet. <https://www.espressif.com/en/products/socs/esp32>, ????
- EUROPEAN CANSAT COMPETITION. Cansat – european space agency. Electronic resource, ESA Education, 2025.

- EUROPEAN SPACE AGENCY. Build your can-sized satellite with cansat 2024–2025. ESA Education, 2024. [https://www.esa.int/Education/Teachers\\_Corner/Build\\_your\\_can-sized\\_satellite\\_with\\_CanSat\\_2024-2025](https://www.esa.int/Education/Teachers_Corner/Build_your_can-sized_satellite_with_CanSat_2024-2025).
- EUROPEAN SPACE AGENCY. European space agency (esa). Sitio web institucional, 2025.
- EXCEL, M. Microsoft excel. <https://www.microsoft.com/en-us/microsoft-365/excel>, ????
- FEDERATION, R. Glonass - global navigation satellite system. <https://www.glonass-iac.ru/en/>, 2024. Accedido en julio de 2025.
- FETTE, I. y MELNIKOV, A. The websocket protocol. <https://datatracker.ietf.org/doc/html/rfc6455>, 2011. IETF RFC 6455.
- GARCIA, E. Técnicas de localización en redes inalámbricas de sensores. 2006.
- HUNTER, J. D. Matplotlib: A 2d graphics environment. <https://matplotlib.org/>, 2003. Accedido en julio de 2025.
- INC., A. Http live streaming (hls). <https://datatracker.ietf.org/doc/html/rfc8216>, 2009–2017 (IETF draft).
- INTERNATIONAL, D. Xbee 802.15.4 rf module. <https://www.digi.com/products/embedded-systems/digi-xbee/rf-modules/2-4-ghz-rf-modules/xbee-802-15-4>, ????
- INTERNATIONAL, D. Xbee rf modules. <https://www.digi.com/xbee>, 2024.
- ISO/IEC. Dynamic adaptive streaming over http (dash) — iso/iec 23009-1:2022. 2022. International Standard for MPEG-DASH.
- JAPAN AEROSPACE EXPLORATION AGENCY. What is cansat? <https://stage.tksc.jaxa.jp/taurus/member/miyazaki/old/e/CanSat.html>, 2003. <https://stage.tksc.jaxa.jp/taurus/member/miyazaki/old/e/CanSat.html>.
- LABS, I. Global navigation satellite system (gnss) and satellite navigation explained. *Inertial Labs White Paper*, 2024.
- LABVIEW NATIONAL INSTRUMENTS. Labview system design software. <https://www.ni.com/en-us/shop/labview.html>, ????
- MACROMEDIA, A. . Real-time messaging protocol (rtmp) specification. <https://rtmp.veriskope.com/docs/spec/>, 2012.
- MICROCHIP TECHNOLOGY. Mcp73871 – li-ion/li-polymer battery charger and power path management controller. <https://ww1.microchip.com/downloads/en/DeviceDoc/MCP73871-Data-Sheet-20002090E.pdf>, 2019.
- NAHALINGAM, K. y KATEHI, L. A review of the recent developments in the fabrication processes of cmos image sensors for smartphones. 2023.
- OFFICE, C. S. N. Beidou navigation satellite system. <http://en.beidou.gov.cn/>, 2024. Accedido en julio de 2025.

- PIVOTAL SOFTWARE, I. Rabbitmq documentation. <https://www.rabbitmq.com/documentation.html>, 2023.
- PYQTGRAPH. Pyqtgraph - scientific graphics and gui library for python. <http://www.pyqtgraph.org/>, ????
- RASPBERRY PI CAMERA MODULE v2. Raspberry pi camera modulev2 – 8mp sony imx219 sensor. <https://www.adafruit.com/product/3099>, 2016.
- RASPBERRY PI FOUNDATION. Raspberry Pi Foundation - Official Website. <https://www.raspberrypi.org/>, ????
- RASPBERRY PI ZERO 2 W. Raspberry Pi Zero 2 W. <https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/>, ????
- RESEARCHGATE. Atmospheric data measurement using can-sat and data logging in ground station - scientific figure on researchgate. [https://www.researchgate.net/figure/A-typical-CanSat-1\\_fig1\\_326505110](https://www.researchgate.net/figure/A-typical-CanSat-1_fig1_326505110), 2018.
- SEMICONDUCTORS, N. Um10204 i<sup>2</sup>c-bus specification and user manual. <https://www.nxp.com/docs/en/user-guide/UM10204.pdf>, 2014.
- SERIALPLOT. Serialplot - real-time plotting software. <https://docs.arduino.cc/software/ide-v2/tutorials/ide-v2-serial-plotter/>, ????
- SHEETS, G. Google sheets. <https://www.google.com/sheets/about/>, ????
- SKYBRARY. International standard atmosphere (isa). <https://skybrary.aero/articles/international-standard-atmosphere-isa>, 2021.
- FOR THE SPACE PROGRAMME, E. U. A. Galileo - european global navigation satellite system. <https://www.euspa.europa.eu/european-space/galileo-services>, 2024. Accedido en julio de 2025.
- SZE, V., BUDAGAVI, M., SULLIVAN, G. y EDITORS. *High Efficiency Video Coding (HEVC): Algorithms and Architectures*. 2014. ISBN 978-3-319-06894-7.
- TECHNOLOGIES, A. *APC220 Radio Data Module Datasheet*, 2010.
- TKINTER. Tkinter — python interface to tcl/tk. <https://docs.python.org/3/library/tkinter.html>, ????
- U.S. GOVERNMENT. Global positioning system (gps). <https://www.gps.gov/systems/gps/>, 2024. Accedido en julio de 2025.
- W3C y IETF. Webrtc: Real-time communication in web browsers. <https://www.w3.org/TR/webrtc/>, 2021.
- WIEGAND, T., SULLIVAN, G., BJØNTEGAARD, G. y LUTHRA, A. Overview of the h.264/avc video coding standard. *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, páginas 560 – 576, 2003.



# Apéndice A

## Descarga e instalación del proyecto

El código desarrollado durante la realización de este trabajo se encuentra disponible en el repositorio: <https://github.com/tronxi/real-time-tracking-system>

Todos los componentes desarrollados para la plataforma de visualización se encuentran dockerizados, por tanto la única herramienta necesaria para desplegar la plataforma es Docker.

El primer paso es clonar el repositorio y acceder a la carpeta docker-compose:

```
↳ ~ git clone https://github.com/tronxi/real-time-tracking-system.git
Cloning into 'real-time-tracking-system'...
remote: Enumerating objects: 1347, done.
remote: Counting objects: 100% (27/27), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 1347 (delta 5), reused 24 (delta 5), pack-reused 1320 (from 2)
Receiving objects: 100% (1347/1347), 33.68 MiB | 2.14 MiB/s, done.
Resolving deltas: 100% (681/681), done.
↳ ~ cd real-time-tracking-system/docker-compose
↳ docker-compose git:(master) █
```

Figura A.1: Clonado del repositorio

A continuación, se debe ejecutar el siguiente comando para construir y lanzar todos los servicios:

```
docker compose up -d --build
```

Este comando creará los contenedores necesarios utilizando los `Dockerfile` definidos en cada componente. para verificar que se han desplegado correctamente, puede utilizarse:

```
docker ps
```

CONTAINER ID	IMAGE	NAMES	COMMAND	CREATED	STATUS	PORTS
110ae93d58e9c	postgres:14.5		"docker-entrypoint.s..."	32 seconds ago	Up 31 seconds	0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp
d7a1b6d54f9	tronci/rabbit-stomp	consat-postgresql	"docker-entrypoint.s..."	32 seconds ago	Up 31 seconds	0.0.0.0:5672->5672/tcp, [::]:5672->5672/tcp, 0.0.0.0:15672->15672/tcp, [::]:15672->15672/tcp, 0.0.0.0:6161
3->61613/tcp, [::]:61613->61613/tcp		rocket-rabbit-stomp	"docker-entrypoint.s..."	32 seconds ago	Up 31 seconds	0.0.0.0:5672->5672/tcp, [::]:5672->5672/tcp, 0.0.0.0:15672->15672/tcp, [::]:15672->15672/tcp, 0.0.0.0:6161
44e8100661b	dashboard-backend:0.0.6-SNAPSHOT	dashboard-backend	"./usr/local/bin/mvn --..."	32 seconds ago	Up 31 seconds	0.0.0.0:8070->8080/tcp, [::]:8070->8080/tcp
85b0c532f046	dashboard-app:latest	dashboard-backend	"./docker-entrypoint..."	32 seconds ago	Up 31 seconds	0.0.0.0:80->80/tcp, [::]:80->80/tcp
9bed636282c9	alqutami/rtmp-hls	dashboard-app	"nginx -g 'daemon of..."	32 seconds ago	Up 31 seconds	0.0.0.0:1935->1935/tcp, [::]:1935->1935/tcp, 0.0.0.0:8080->8080/tcp, [::]:8080->8080/tcp
		rtmp-hls				

Figura A.2: Ejecución del comando docker ps

Como podemos ver en la salida del comando tendremos los servicios levantados en los siguientes puertos:

- **Interfaz web:** <http://localhost>
- **Backend:** <http://localhost:8070>
- **PostgreSQL:** [localhost:5432](http://localhost:5432)
- **RabbitMQ:** <http://localhost:15672>
- **Servidor RTMP:** <http://localhost:8080>

Las credenciales y contraseñas necesarias para los distintos servicios están definidas como variables de entorno en el fichero `.env`, ubicado en la carpeta `docker-compose/`. Estas pueden modificarse para adaptarlas a las necesidades de cada proyecto.

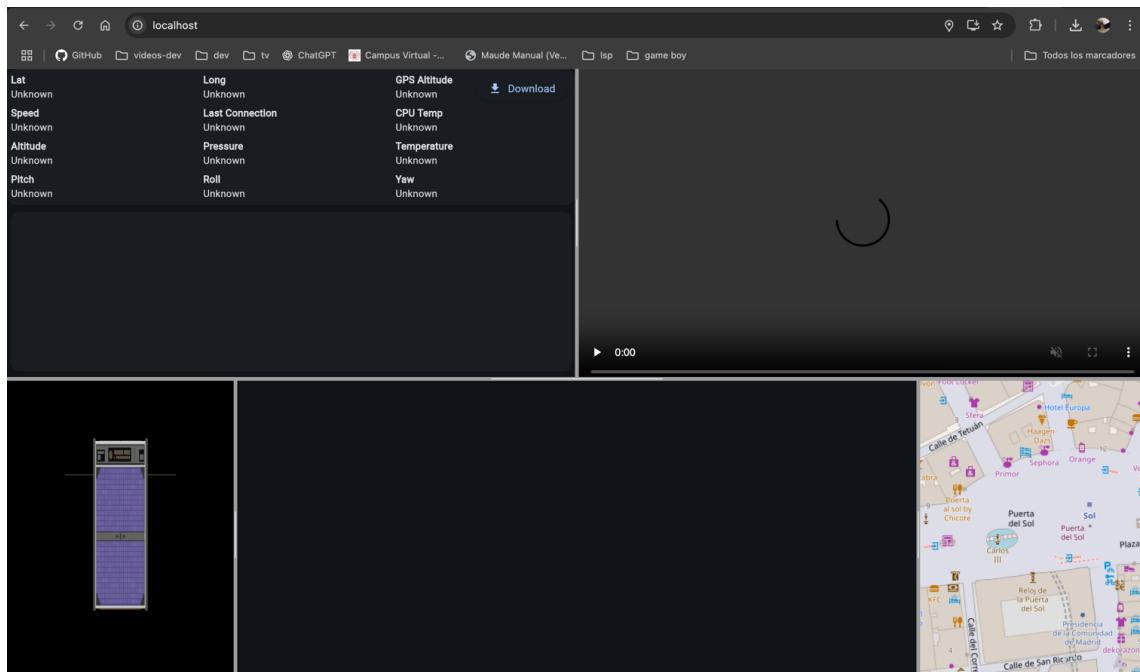


Figura A.3: Interfaz Web corriendo en el puerto 80

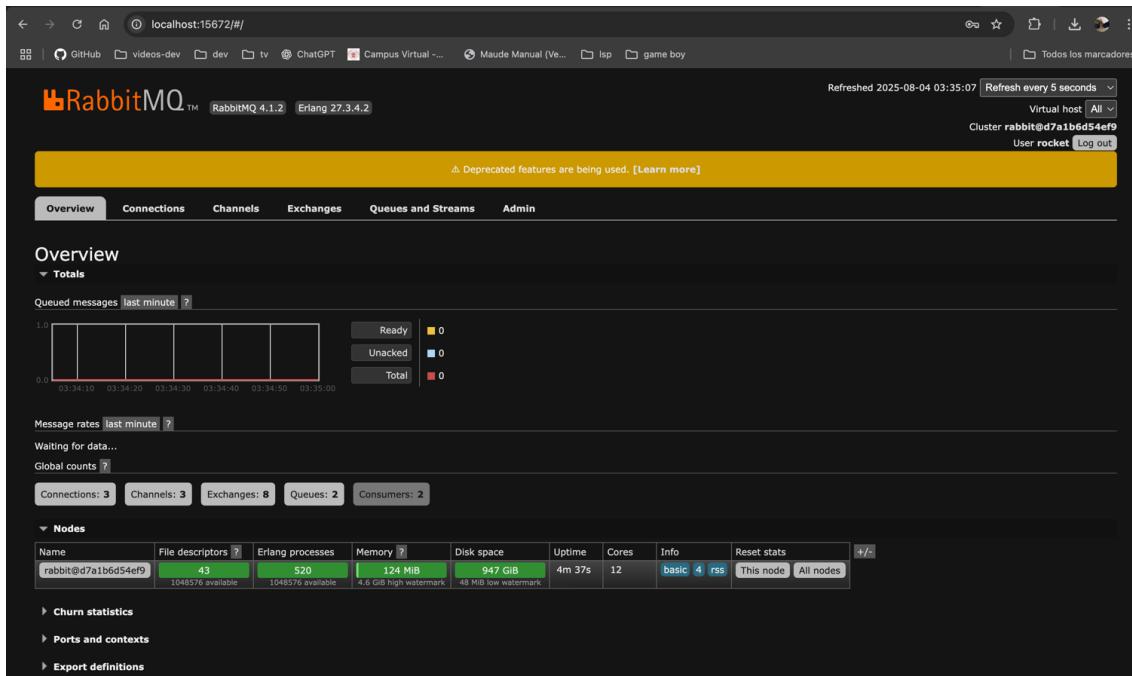


Figura A.4: Interfaz RabbitMQ corriendo en el puerto 15672

