

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

Sergio García Sánchez — Máster en Ingeniería Informática (UCM)

Director: Adrián Riesco Rodríguez



UNIVERSIDAD
COMPLUTENSE
MADRID

Índice

1. Introduction

2. Objectives

3. Fundamentos y estado del arte

4. Diseño y arquitectura

5. Validación y resultados

6. Conclusions

7. Future work

8. Q&A

1. Introduction

This project presents the **design and implementation** of a complete system for:

- Building a **CanSat-like device** with sensors, GPS and camera
- Sending data in real time via **WiFi or radio**
- Displaying telemetry, position, attitude, and video in a **reusable web interface**

What is a CanSat?

- Miniature satellite with the approximate size of a soda can (66 mm diameter × 115 mm height)
- Equipped with sensors, onboard computer, power supply, radio system, and parachute
- Launched from rockets or drones to simulate a space mission, typically from 500 – 1000 m altitude
- Widely used for educational purposes and in competitions (ESA CanSat, NASA CanSat)



Motivation

- Existing CanSat projects focus mainly on hardware and electronics
- Data visualization is often ad-hoc, not reusable
- Lack of an easy-to-deploy **modular web platform** for telemetry and video

2. Objectives – CanSat

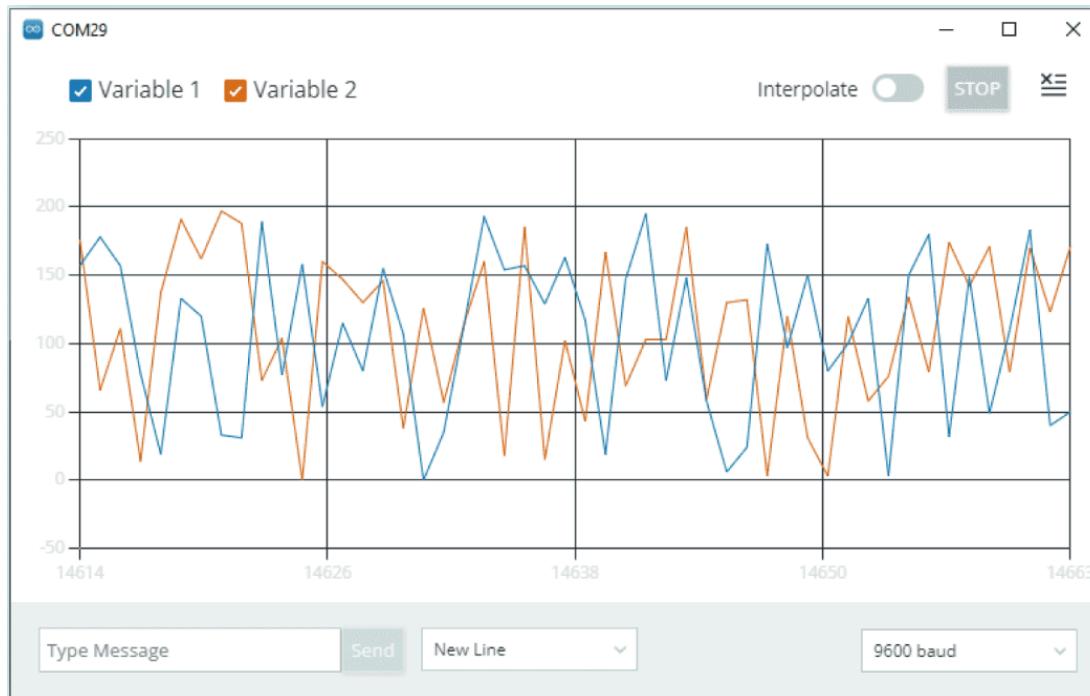
- Build a **CanSat** (66 × 115 mm, 300–350 g) with:
 - GPS receiver
 - Camera for live video
 - Barometric sensor for pressure/altitude
 - IMU for attitude
 - Battery with solar charging option
- Send data in real time:
 - **WiFi** when available
 - **LoRa radio** as fallback (with ground station)

2. Objectives – Web Platform

- Provide a **modular, reusable platform** for:
 - Live telemetry and last values
 - Real-time charts of sensor data
 - GNSS position on a map
 - 3D attitude model
 - Live video streaming
 - Data export for analysis

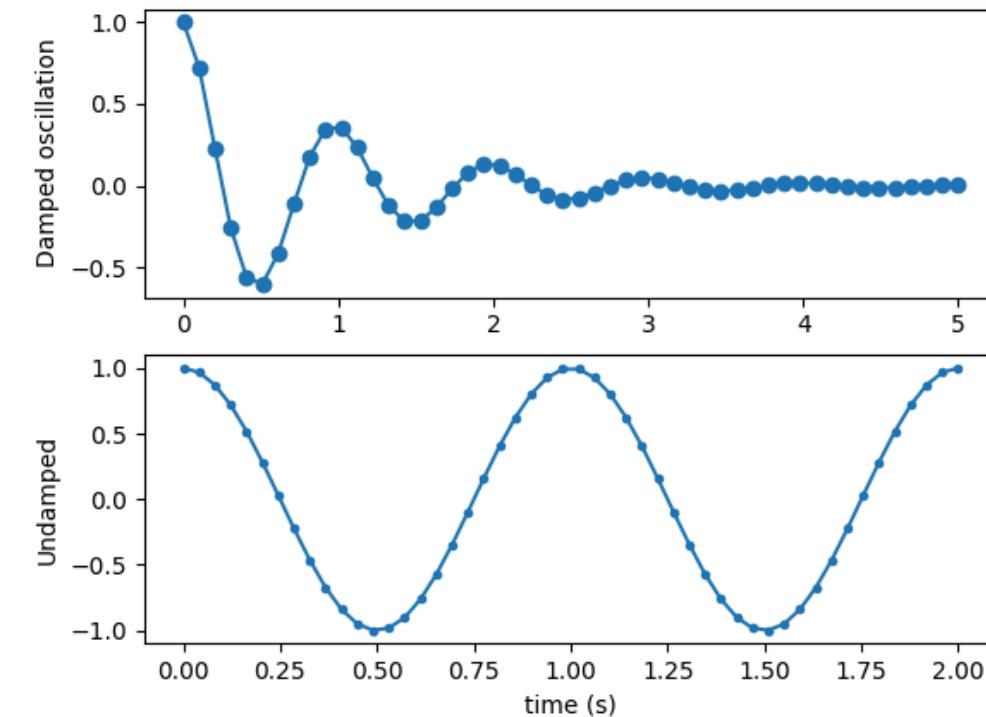
3. Fundamentos - Herramientas de visualización

SerialPlot: visor de puerto serie para ver y graficar datos en tiempo real desde Arduino IDE.

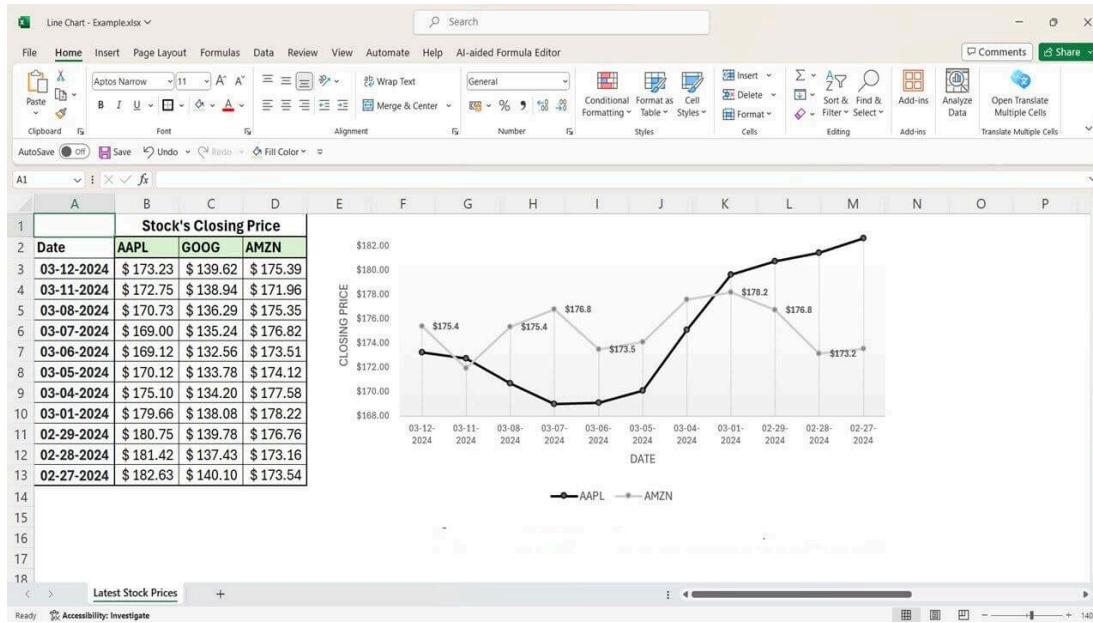


Matplotlib / PyQtGraph: librerías de Python para generar gráficos.

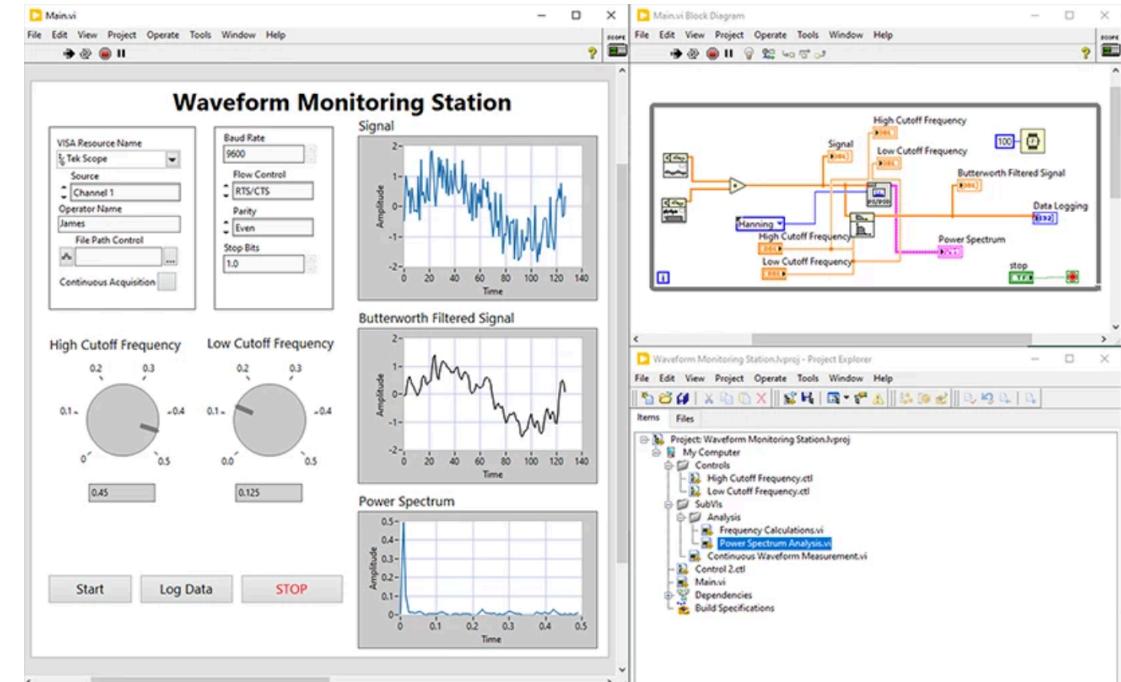
A tale of 2 subplots



Excel / Google Sheets: hojas de cálculo para calcular y graficar datos, no soporta streaming.



LabVIEW: entorno gráfico de National Instruments para adquisición de datos y control.



3. Fundamentos - Microcontroladores

Característica	Raspberry Pi Zero 2 W	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 ² 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 €

3. Fundamentos – Interfaces de comunicación

Característica	SPI	I ² C	UART
Tipo de comunicación	Síncrona	Síncrona	Asíncrona
Arquitectura	Primario–secundario	Primario–secundario	Punto a punto
Nº de líneas	4 (CLK, CS, MOSI, MISO)	2 (SDA, SCL)	2 (TX, RX)
Dúplex	Full	Half	Full
Nº de dispositivos	1 primario, varios secundarios	1 primario, varios secundarios	Solo dos
Velocidad típica	1–10 Mbps	100 kbit/s – 3.4 Mbps	9.6 kbit/s – 3 Mbps
Control de dirección	Señal CS por secundario	Dirección en protocolo	No necesario

3. Fundamentos – Comunicación por radiofrecuencia

Característica	LoRa	XBee (802.15.4)	APC220
Frecuencia	433 / 868 / 915 MHz	2.4 GHz	433 MHz
Modulación	CSS (Chirp Spread)	DSSS + O-QPSK	GFSK
Velocidad de datos	0.3–27 kbps	250 kbps	1.2–19.2 kbps
Alcance típico	hasta 15 km	30–300 m	hasta 1 km
Corrección errores	FEC (CR 4/5–4/8)	No especificado	FEC + AGC
Interfaz MCU	UART	UART	UART

3. Fundamentos – Sensor de presión barométrica

- Mide **presión atmosférica** → se calcula altitud usando el modelo ISA
- A mayor altura, menor presión (relación con densidad del aire)
- Factores que afectan la precisión: temperatura, humedad, viento
- Usado para estimar **altura relativa durante el vuelo**

Ejemplos de sensores comunes:

Sensores de presión / altitud

Sensor	Rango de presión	Precisión	Interfaz	Consumo	Precio
BMP388	300–1250 hPa	±8 Pa (±0,66 m)	I ² C / SPI	3.4 µA	~3,50 €
BME280	300–1100 hPa	±12 Pa (±1 m)	I ² C / SPI	2.7 µA	~4,00 €
MPL3115A2	50–1100 hPa	±0,04 hPa (±0,3 m)	I ² C	40 µA	~6,00 €

3. Fundamentos – IMU (Inertial Measurement Unit)

- Mide **aceleración** (3 ejes) y **velocidad angular** (3 ejes)
- Algunos modelos incluyen **magnetómetro** para calcular rumbo
- Permite conocer la **orientación** en términos de pitch, yaw y roll
- Modelos avanzados incluyen procesador interno para fusión sensorial

Sensores IMU:

Sensor	Componentes	Salida de orientación	Interfaz	Consumo	Precio
MPU6050	Acelerómetro + Giroscopio	No (requiere fusión externa)	I ² C	3.9 mA	~1,50 €
BNO055	Acelerómetro + Giroscopio + Magnetómetro	Sí (fusión interna)	I ² C / UART	12 mA	~9,00 €
BNO085	Acelerómetro + Giroscopio + Magnetómetro	Sí (mayor precisión)	I ² C / UART / SPI	3.5 mA	~14,00 €

3. Fundamentos – GNSS

- Sistemas: **GPS, Galileo, GLONASS, BeiDou**
- Un receptor necesita ≥ 3 satélites para calcular posición (trilateración)
- Variables: precisión, frecuencia de actualización, constelaciones soportadas

Comparativa de módulos GNSS:

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

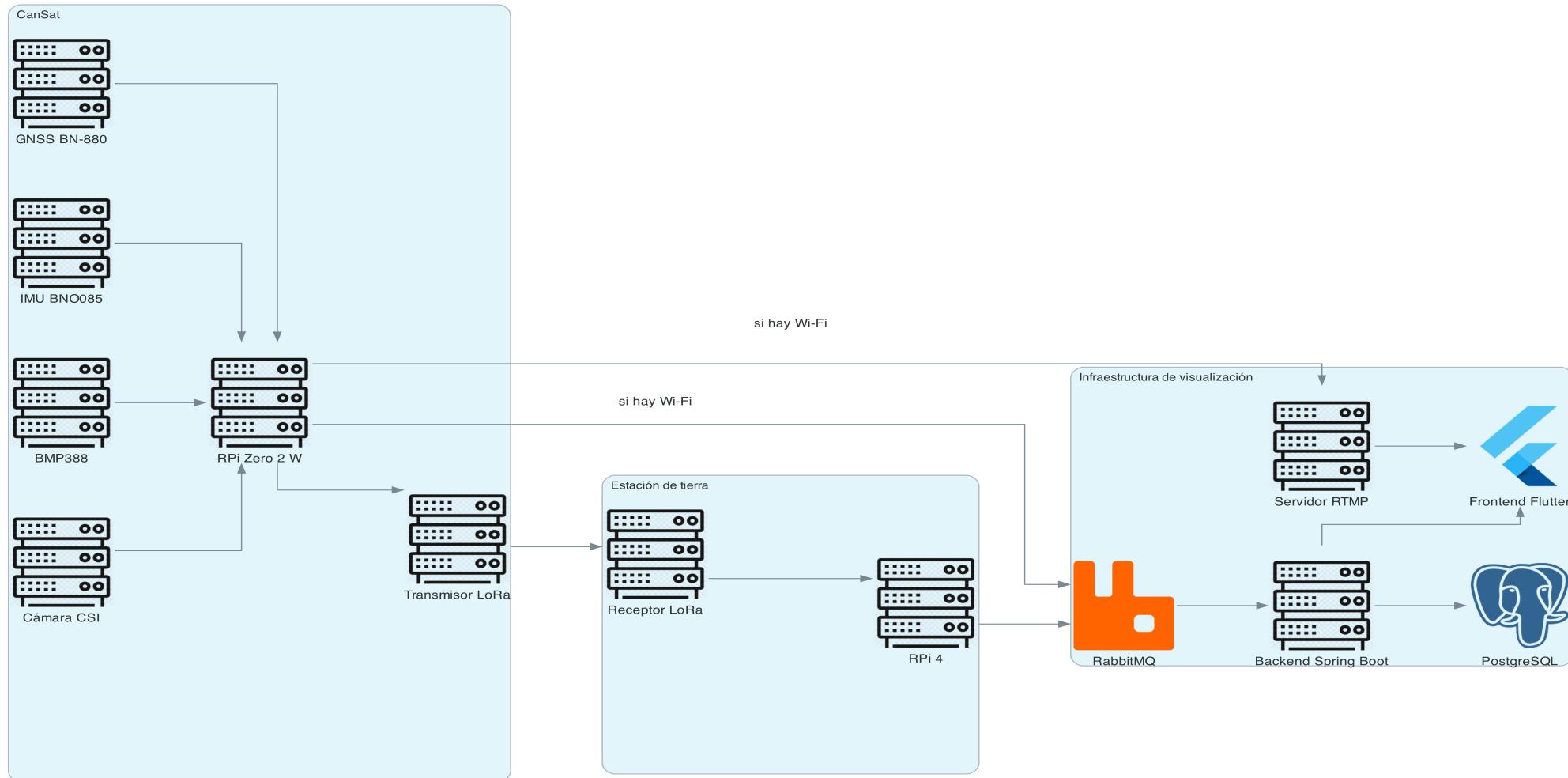
4. Diseño e implementación – Hardware

- **Unidad de procesamiento:** Raspberry Pi Zero 2 W
- **Sensores:**
 - BMP388 (presión y temperatura)
 - BNO085 (IMU con fusión sensorial)
 - BN-880 (GNSS multi-constelación + brújula)
- **Cámara CSI:** Raspberry Pi Camera Module v2
- **Transmisión:** WiFi si hay red disponible, LoRa E32-900T20D si no
- **Alimentación:** batería 18650 + cargador MCP73871 + boost converter a 5V
- **Estación de tierra:** Raspberry Pi 4 + receptor LoRa

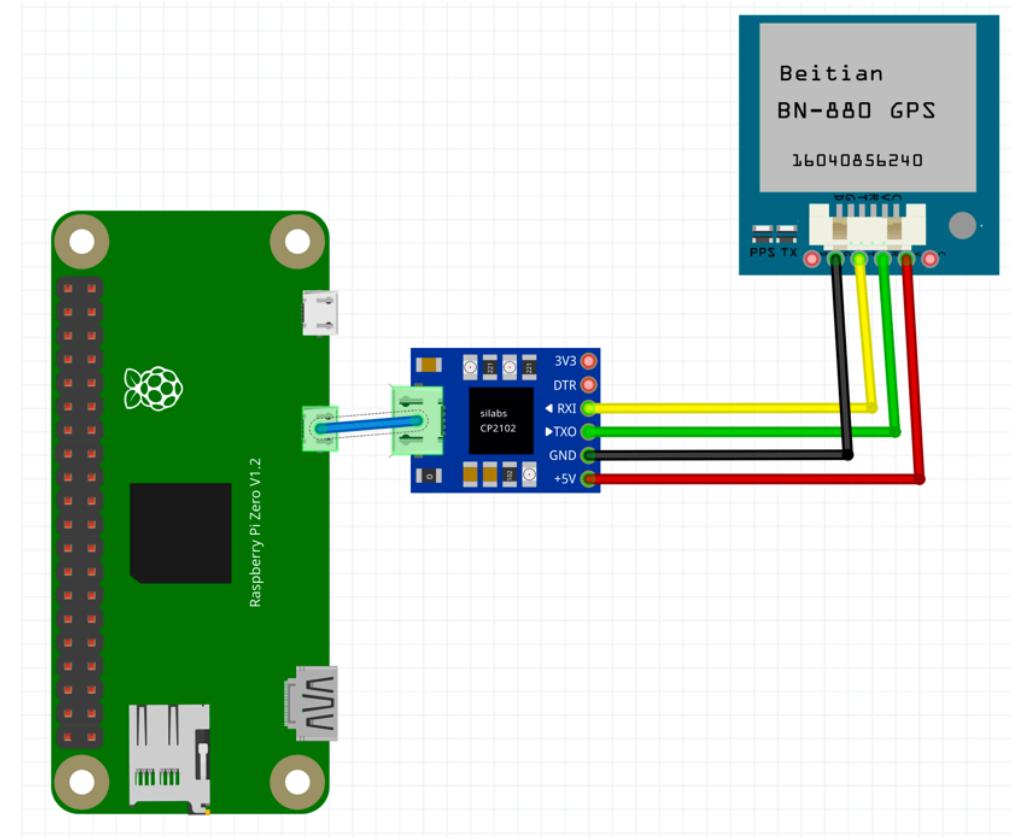
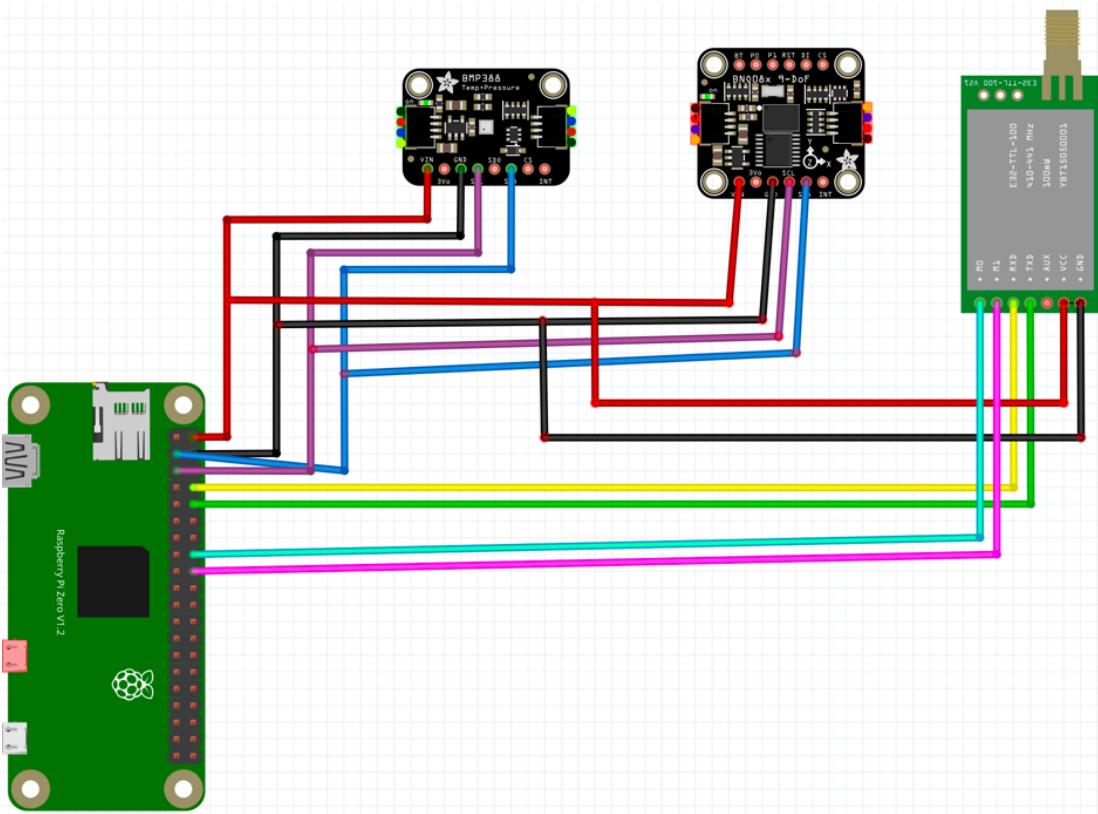
4. Selección de tecnologías – Software

- **Software embebido:**
 - Scripts en Python para lectura de sensores, GNSS, captura de vídeo y envío de datos
- **Backend:**
 - Spring Boot (Java) + API REST
 - RabbitMQ (AMQP) para mensajería de eventos
 - PostgreSQL para persistencia
- **Frontend:**
 - Desarrollado en **Flutter**
 - Comunicación en tiempo real mediante WebSocket

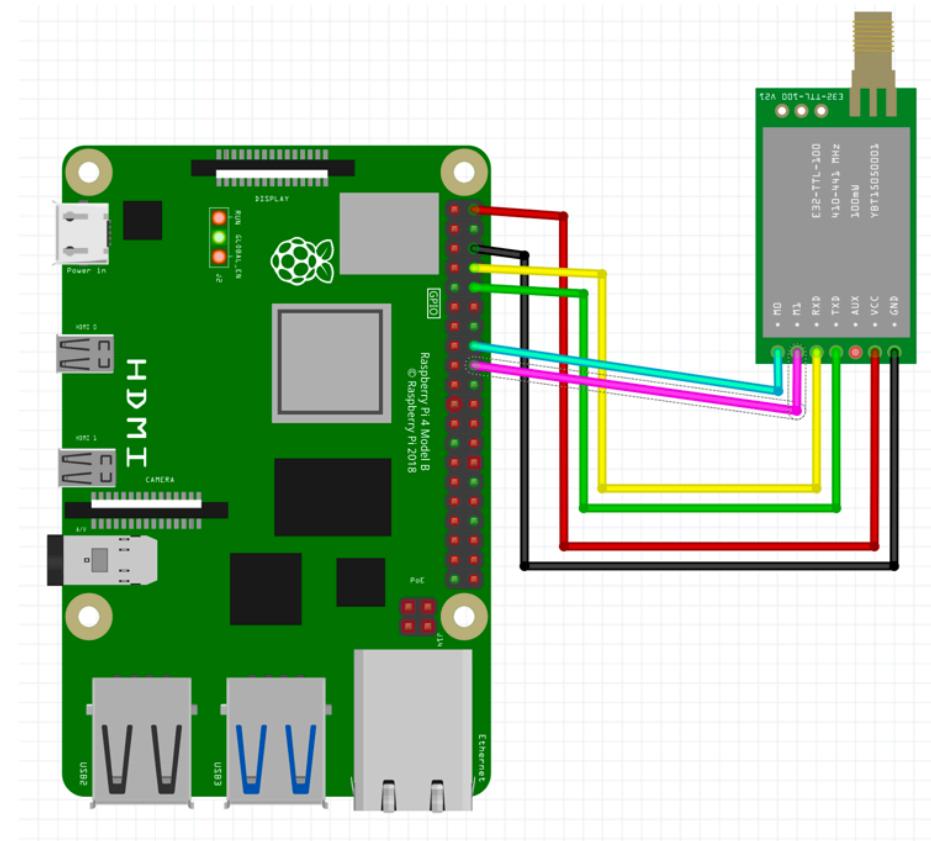
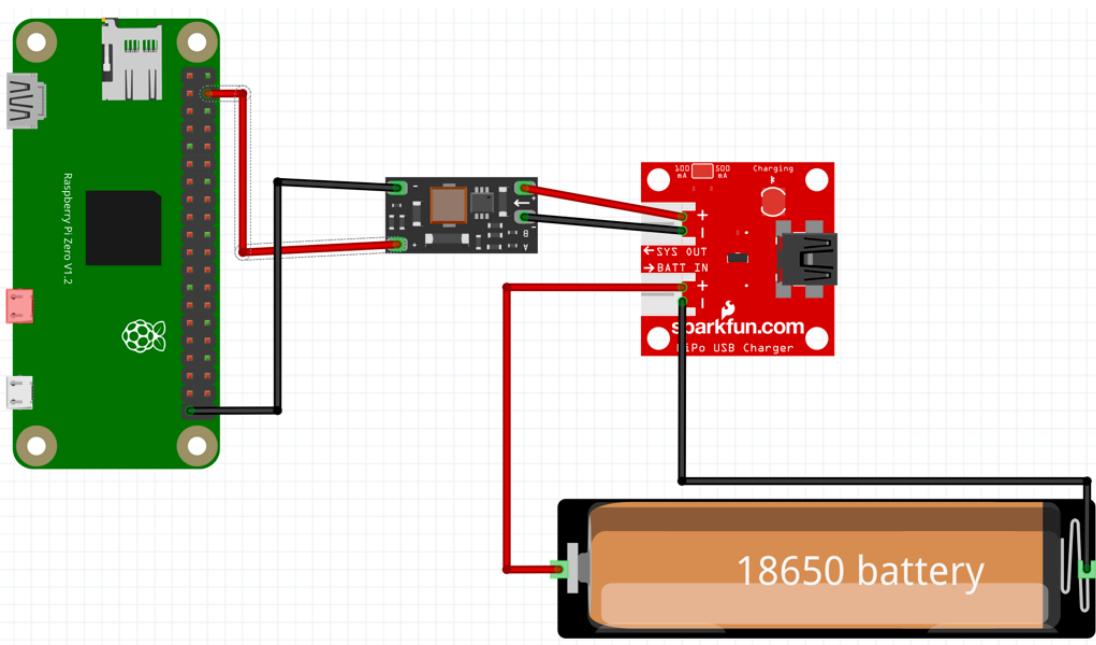
4. Selección de tecnologías – Arquitectura general



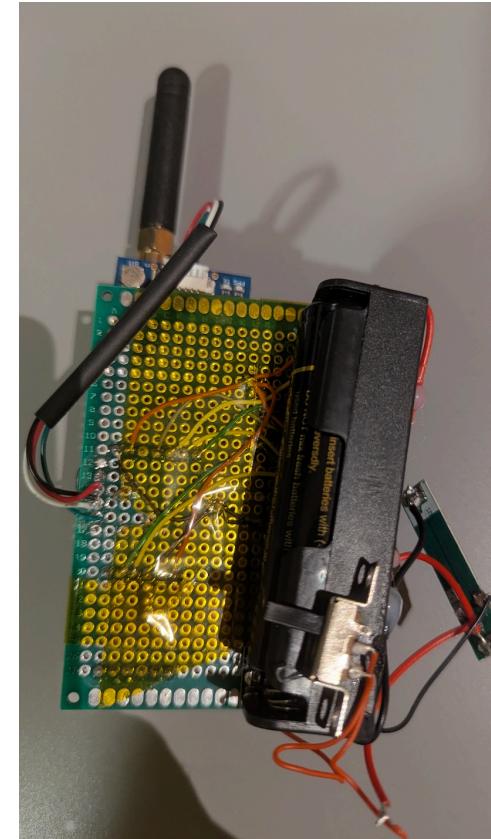
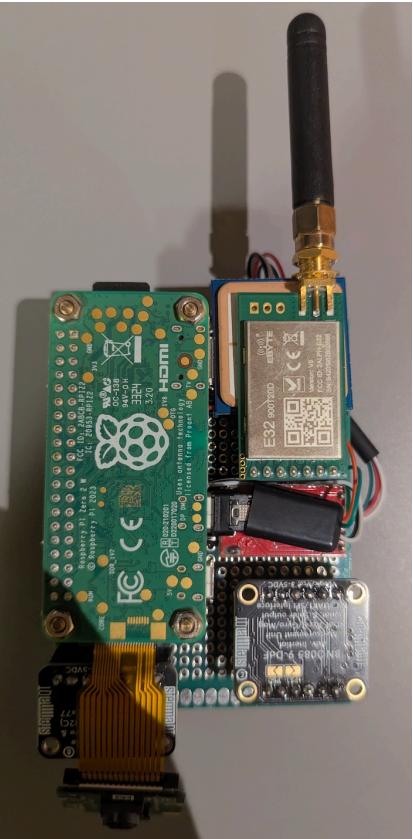
4. Selección de tecnologías – Conexión



4. Selección de tecnologías – Conexión



4. Selección de tecnologías – Conexión



4. Selección de tecnologías – Interfaz

Lat	Long	GPS Altitude	Download
Unknown	Unknown	Unknown	
Speed	Last Connection	CPU Temp	
Unknown	2025-08-03T04:16:51.642376	55.31	
Altitude	Pressure	Temperature	
0.38	942.75	37.25	
Pitch	Roll	Yaw	
4.439005091550756	100.77675157168527	178.35076118680433	

178.36456678270093, pressure: 942.66, temperature: 37.24, cpuTemperature: 54.77}}
{type: TM, datetime: 2025-08-03T04:16:44.06412, payload: {altitude: 0.37, roll: 100.78778027685442, pitch: 4.443648622948188, yaw: 178.3565844623096, pressure: 942.7, temperature: 37.24, cpuTemperature: 54.77}}
{type: TM, datetime: 2025-08-03T04:16:44.305242, payload: {altitude: 0.46, roll: 100.78800230770547, pitch: 4.442711350862725, yaw: 178.36644850165376, pressure: 942.71, temperature: 37.24, cpuTemperature: 54.77}}
{type: TM, datetime: 2025-08-03T04:16:44.549438, payload: {altitude: -0.44, roll: 100.78778027685442, pitch: 4.443648622948188, yaw: 178.3565844623096, pressure: 942.56, temperature: 37.25, cpuTemperature: 54.77}}
{type: TM, datetime: 2025-08-03T04:16:51.642376, payload: {altitude: 0.38, roll: 100.77675157168527, pitch: 4.439005091550756, yaw:



5. Pruebas – Integración con datos simulados

- **Objetivo:** validar backend y frontend sin hardware
- **Telemetría simulada (Python + RabbitMQ):**
 - Publicación periódica de eventos JSON (altitud, temperatura, GNSS, actitud)
 - Verificación de ingesta, persistencia y actualización **en tiempo real** (métricas, gráficas, mapa)
- **Vídeo simulado (Python + FFmpeg → RTMP):**
 - Generación de frames sintéticos y envío a **servidor RTMP**
 - Reproducción en el **frontend** con baja latencia

5. Pruebas – Sistema real (CanSat + Estación + Servicios)

- **Transmisión por WiFi:**

Eventos → RabbitMQ → PostgreSQL → WebSocket → Frontend

Latencia < 1 s desde la generación a la visualización

- **Transmisión por LoRa (868 MHz):**

Recepción estable en campo abierto **hasta > 1 km**

Pérdidas ocasionales a distancias superiores (acorde al módulo empleado)

- **Autonomía energética:** ≈ 2 h de operación continua con telemetría + vídeo

- **Persistencia y exportación:**

Datos en **PostgreSQL** y descarga en **JSONL** vía API

6. Conclusions

- **Goal achieved:** reusable, hardware-independent platform for real-time CanSat data visualization.
- **Modular architecture:** data capture, transmission, backend, frontend → independent changes; easier maintenance & reuse.
- **Validation:** built a real CanSat; hardware-independent; adaptable to other sensors/configurations.
- **Video challenge (Pi Zero 2 W):** unstable → 640×480 @ 15 fps + hardware-accelerated FFmpeg → smooth, continuous video.
- **I/O challenge:** single UART used by LoRa → added USB–UART for GNSS; physically modified the adapter to fit the CanSat

7. Future Work

- Add more **real-time charts** and custom visualizations
- Implement **telecommanding system** to remotely control sensors and configuration
- Add **authentication** to protect sensitive data
- Support monitoring of **multiple CanSats** simultaneously

Preguntas

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

Sergio García Sánchez

Gracias por su atención