

# **CanSat: sistema de adquisición y datos en tiempo real**

Sergio García Sánchez



UNIVERSIDAD  
**COMPLUTENSE**  
MADRID

# **Índice**

**1. Introducción**

**2. Objetivos**

**3. Fundamentos y estado del arte**

**4. Diseño y arquitectura**

**5. Validación y resultados**

**6. Conclusiones**

**7. Trabajo futuro**

**8. Preguntas**

# 1. Introducción

Este proyecto presenta el **diseño e implementación** de un sistema completo orientado a:

- Desarrollar una **arquitectura reutilizable** para dispositivos tipo CanSat
- Integrar adquisición de datos (sensores, GNSS y cámara) en **distintas plataformas hardware**
- Enviar datos en tiempo real mediante **WiFi o radio**
- Visualizar telemetría, posición, actitud y vídeo en una **interfaz web reutilizable**

# ¿Qué es un CanSat?

- Satélite en miniatura con un tamaño aproximado al de una lata de refresco (66 mm de diámetro × 115 mm de altura)
- Equipado con sensores, ordenador a bordo, sistema de alimentación, comunicaciones por radio y paracaídas
- Lanzado desde cohetes o drones para simular una misión espacial, normalmente desde altitudes de 500 a 1000 m
- Ampliamente utilizado con fines educativos y en competiciones (ESA CanSat, NASA CanSat)



## Motivación

- Muchos proyectos CanSat se centran principalmente en el **hardware y la electrónica**, dejando el sistema software en un segundo plano
- La visualización de datos suele ser **ad hoc**, poco estructurada y difícil de reutilizar entre proyectos
- No existe una **arquitectura común**, modular y fácilmente desplegable, para la gestión de telemetría y vídeo en tiempo real

## 2. Objetivos – CanSat

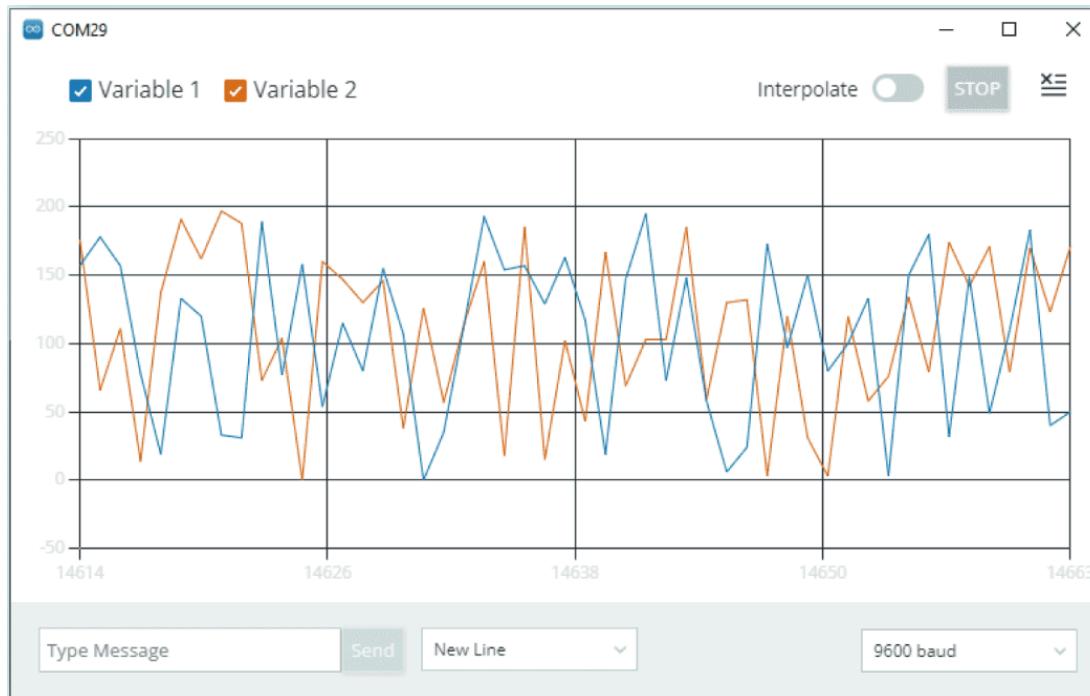
- Diseñar e implementar **dos CanSat funcionales**, basados en:
  - **ESP32**, orientado a bajo consumo y telemetría
  - **Raspberry Pi**, con mayor capacidad de proceso y soporte de vídeo
- Integrar adquisición de datos mediante:
  - Receptor GNSS
  - Sensores ambientales (presión/altitud)
  - IMU para estimación de la actitud
  - Cámara para vídeo en tiempo real
- Enviar los datos en tiempo real mediante:
  - **WiFi**, cuando hay red disponible
  - **Radio LoRa** como mecanismo alternativo, junto con una estación de tierra

## 2. Objetivos – Plataforma web

- Proporcionar una **plataforma modular y reutilizable** que permita:
  - Visualización de telemetría en tiempo real y últimos valores
  - Gráficas en tiempo real de los datos de los sensores
  - Posición GNSS sobre un mapa
  - Modelo 3D de la actitud
  - Streaming de vídeo en tiempo real
  - Exportación de datos para su análisis posterior

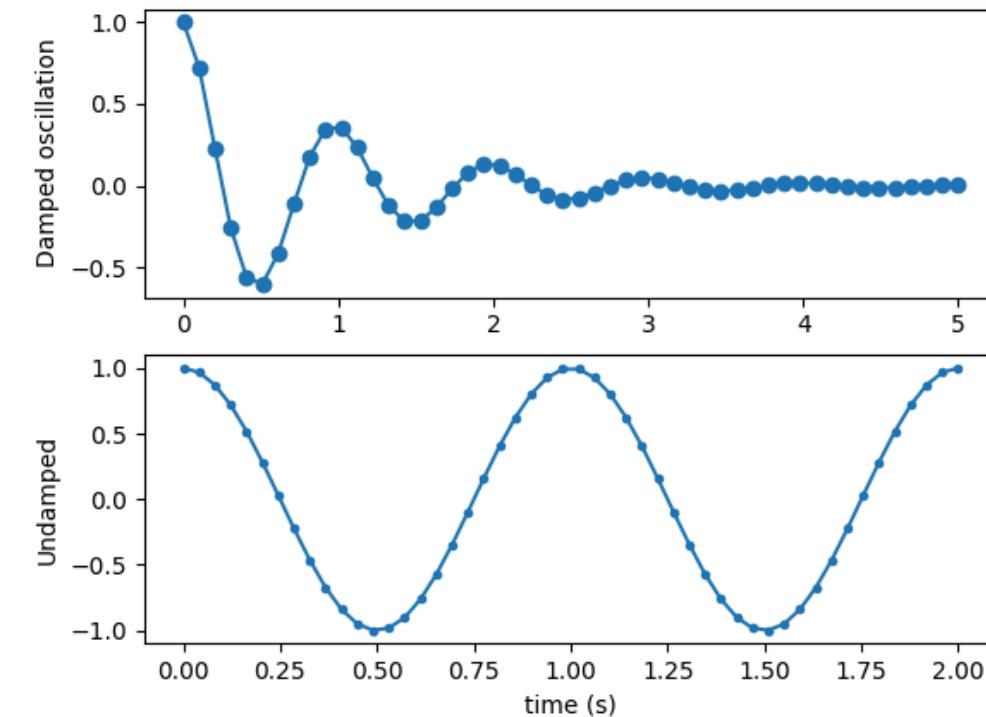
### 3. Fundamentos - Herramientas de visualización

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

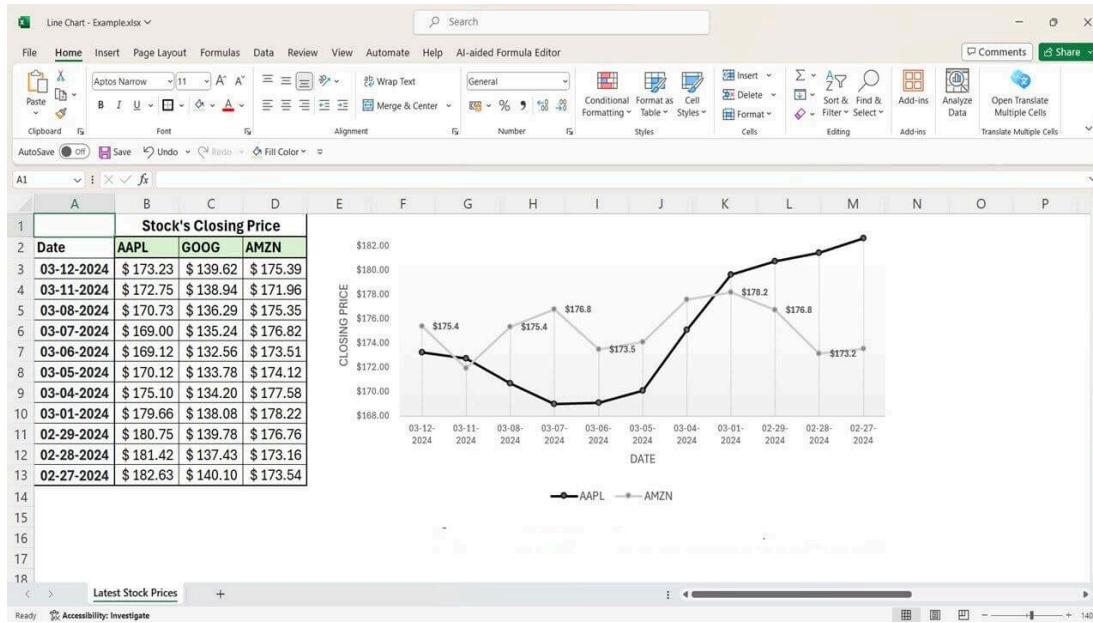


**Matplotlib / PyQtGraph:** librerías de Python para generar gráficos.

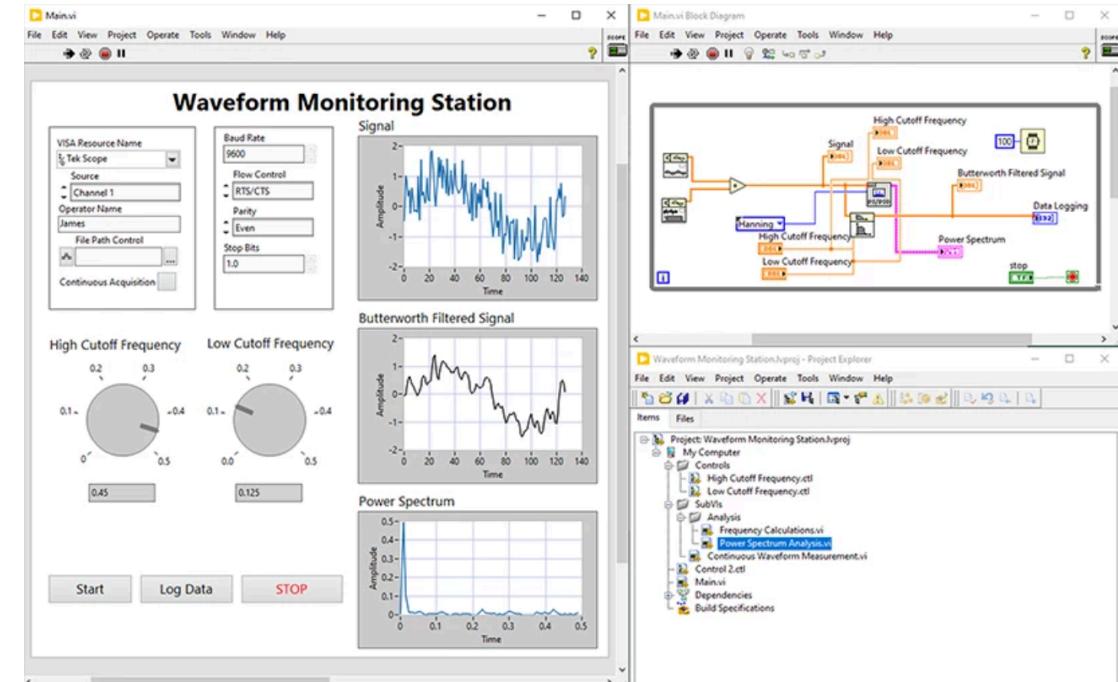
A tale of 2 subplots



**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 <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 €

### 3. Fundamentos – Interfaces de comunicación

Característica	SPI	I <sup>2</sup> 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
<b>Frecuencia</b>	433 / 868 / 915 MHz	2.4 GHz	433 MHz
<b>Modulación</b>	CSS (Chirp Spread)	DSSS + O-QPSK	GFSK
<b>Velocidad de datos</b>	0.3–27 kbps	250 kbps	1.2–19.2 kbps
<b>Alcance típico</b>	hasta 15 km	30–300 m	hasta 1 km
<b>Corrección errores</b>	FEC (CR 4/5–4/8)	No especificado	FEC + AGC
<b>Interfaz MCU</b>	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:

Sensor	Rango de presión	Precisión	Interfaz	Consumo	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 €

### 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 <sup>2</sup> C	3.9 mA	~1,50 €
BNO055	Acelerómetro + Giroscopio + Magnetómetro	Sí (fusión interna)	I <sup>2</sup> C / UART	12 mA	~9,00 €
BNO085	Acelerómetro + Giroscopio + Magnetómetro	Sí (mayor precisión)	I <sup>2</sup> C / UART / SPI	3.5 mA	~14,00 €

### 3. Fundamentos – GNSS

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

#### Comparativa de módulos GNSS:

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

## 4. Diseño e implementación – Hardware

Se han desarrollado dos CanSat usando distintos controladores:

- **CanSat basado en ESP32**
  - Unidad de procesamiento: ESP32
  - Sensores:
    - BMP388 (presión y temperatura)
    - BNO085 (IMU con fusión sensorial)
  - Transmisión: LoRa (SX1278)
- **Estación de tierra**
  - Receptor LoRa (SX1278)

- **CanSat basado en Raspberry Pi**
  - Unidad de procesamiento: Raspberry Pi Zero 2 W
  - Sensores:
    - BMP388 (presión y temperatura)
    - BNO085 (IMU con fusión sensorial)
    - BN-880 (GNSS multi-constelación)
  - Cámara CSI: Raspberry Pi Camera Module v2
  - Transmisión: WiFi y LoRa (E32-900T20D)
  - Alimentación: batería 18650 + cargador MCP73871 + convertidor boost a 5 V
- **Estación de tierra**
  - Raspberry Pi 4
  - Receptor LoRa

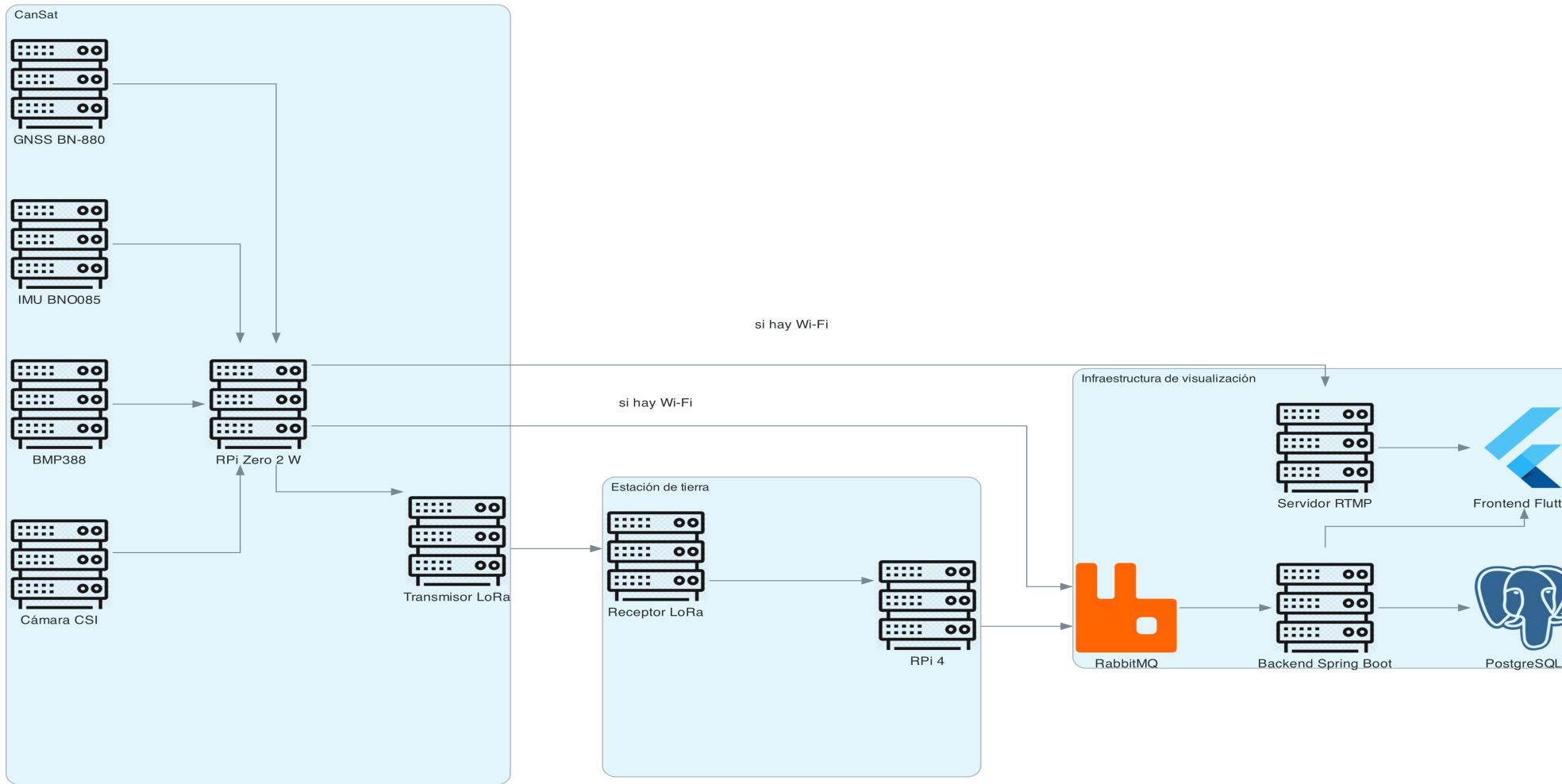
## 4. Diseño e implementación – Hardware

- **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

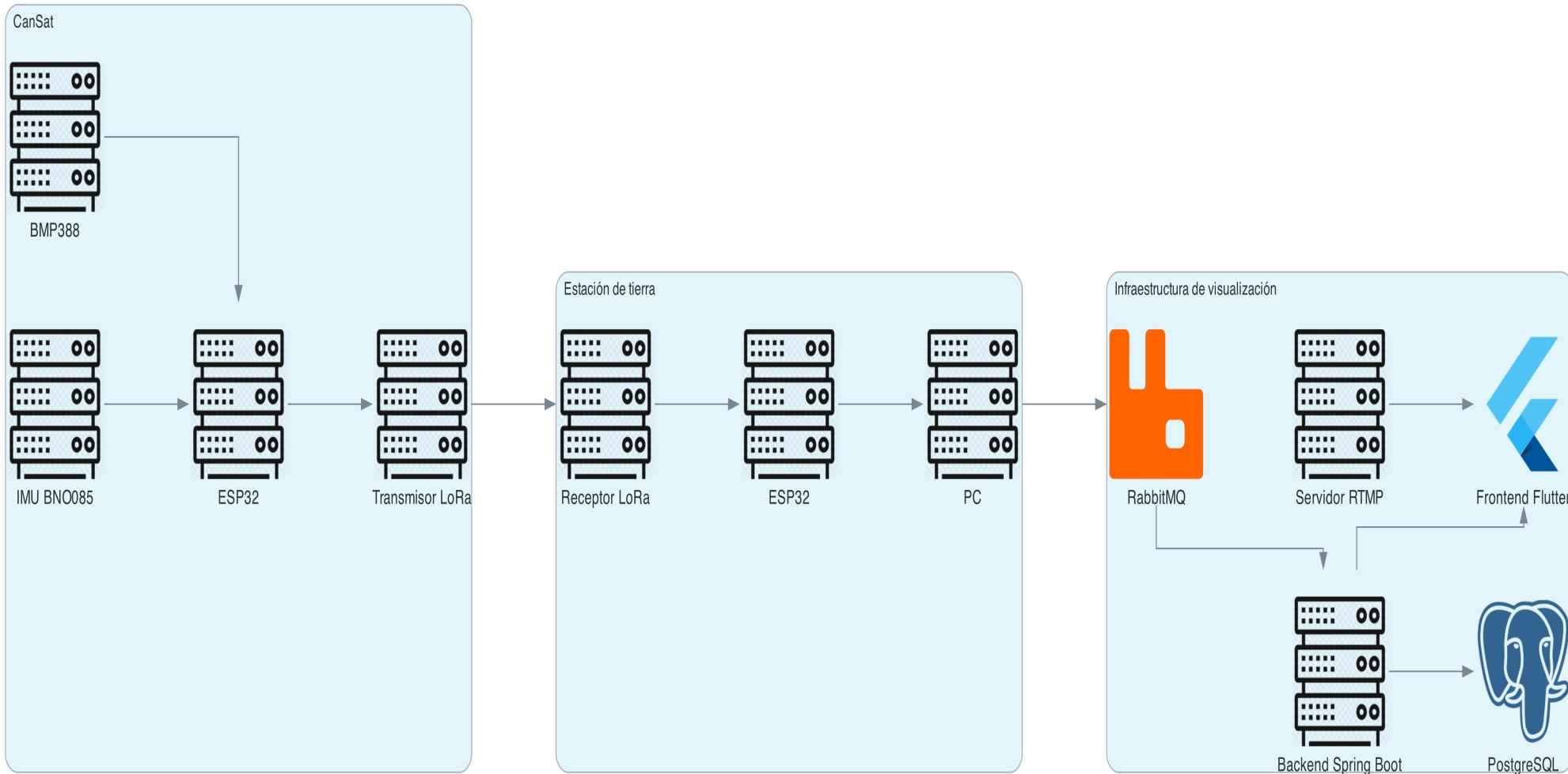
## 4. Selección de tecnologías – Software

- **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

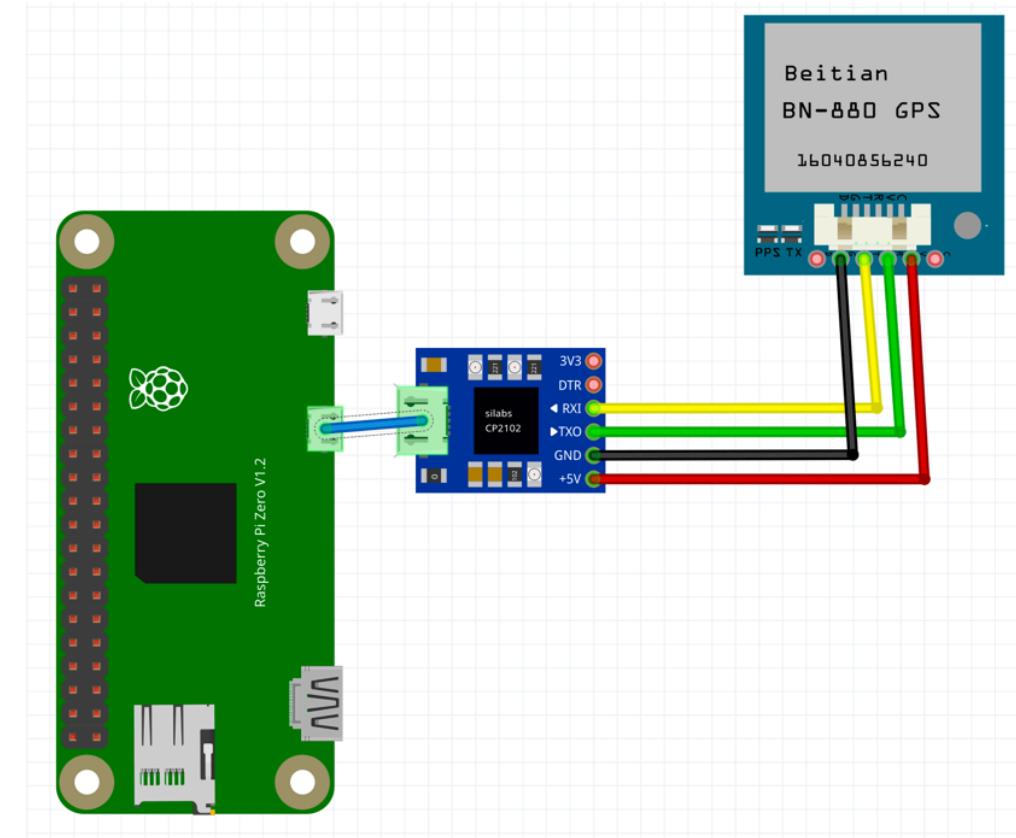
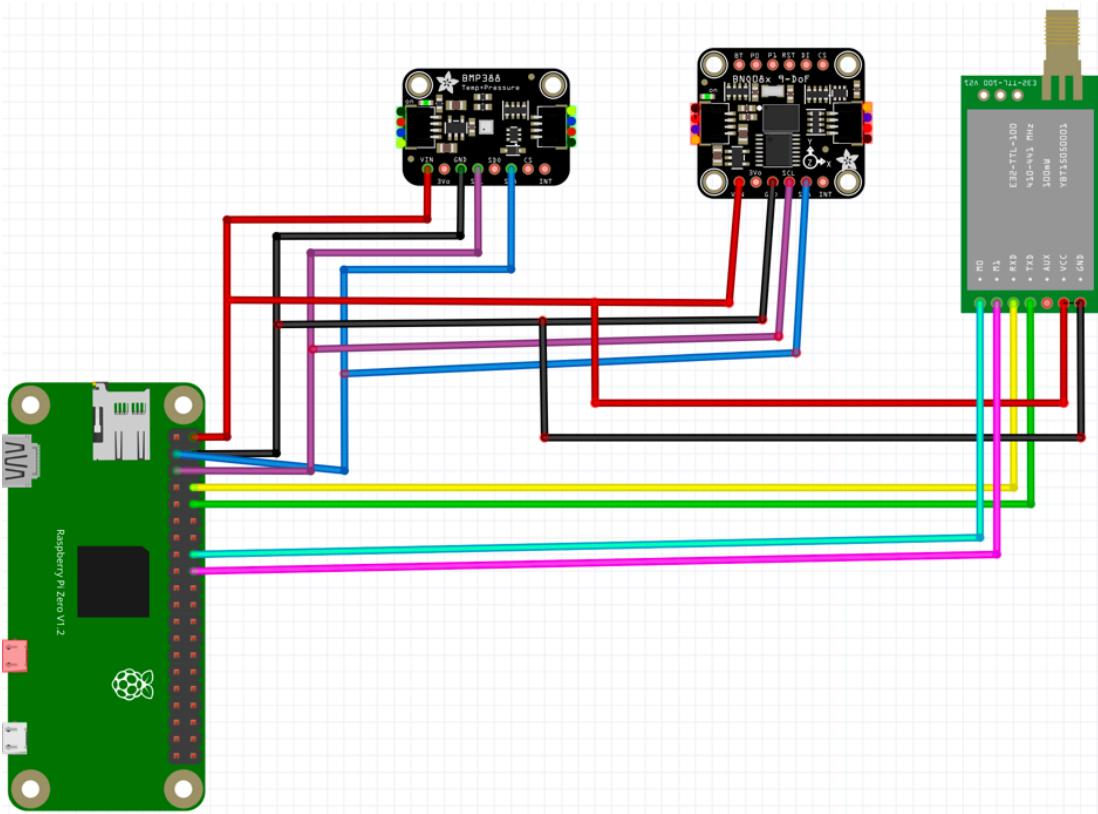
## 4. Selección de tecnologías – Arquitectura general CanSat Raspberry Pi



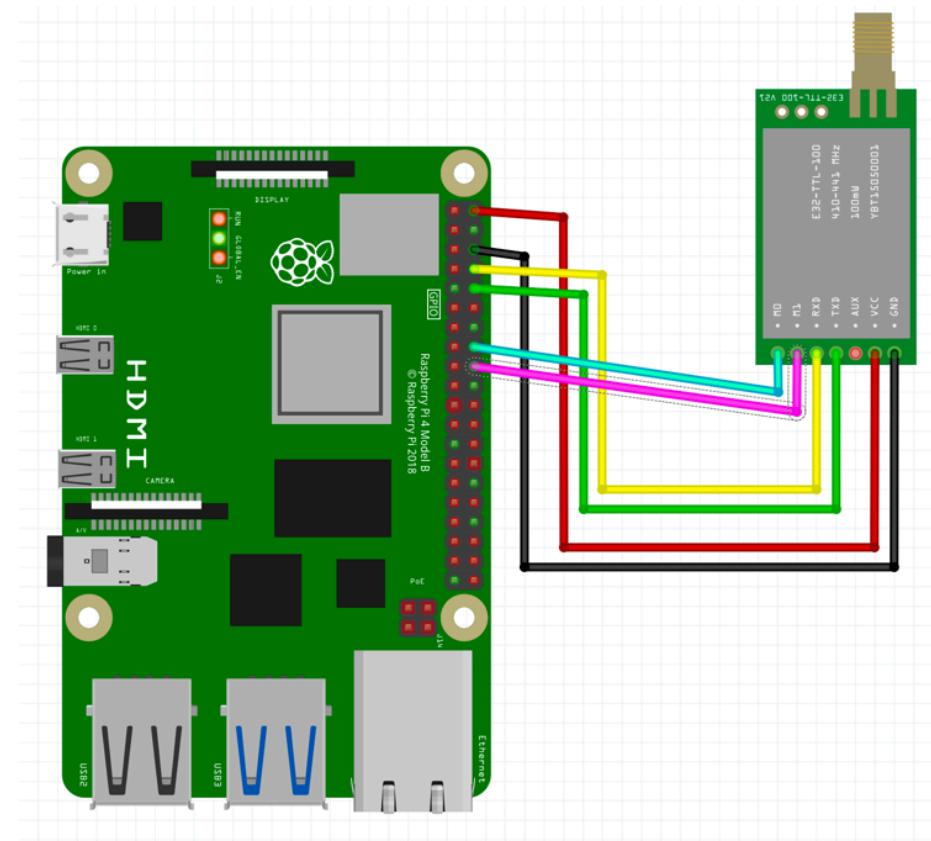
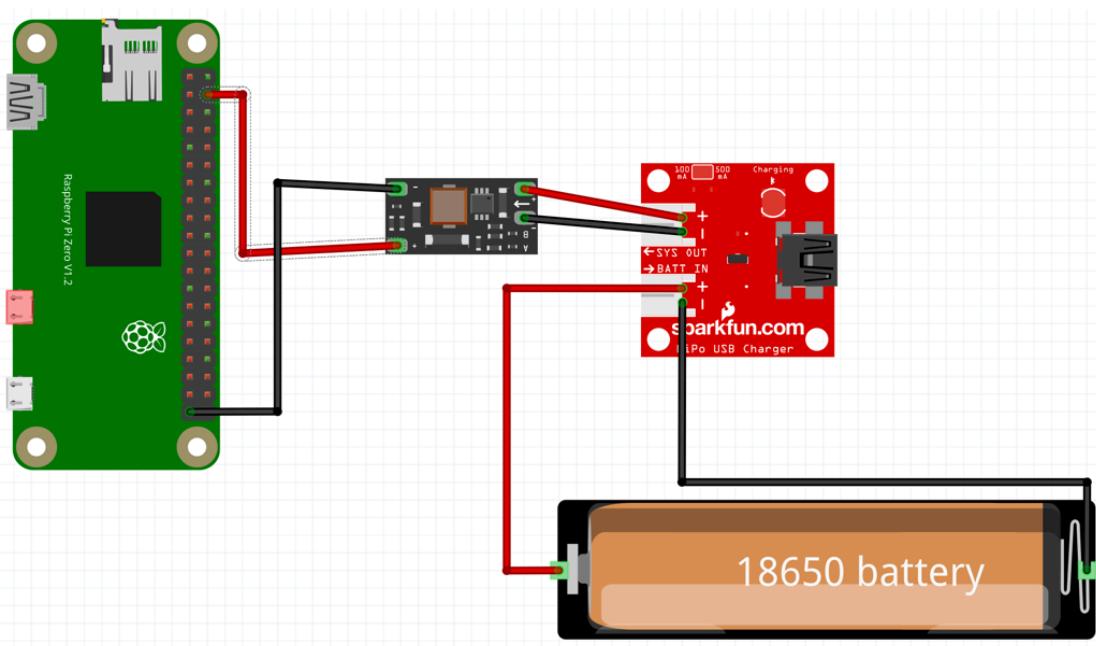
## 4. Selección de tecnologías – Arquitectura general CanSat ESP32



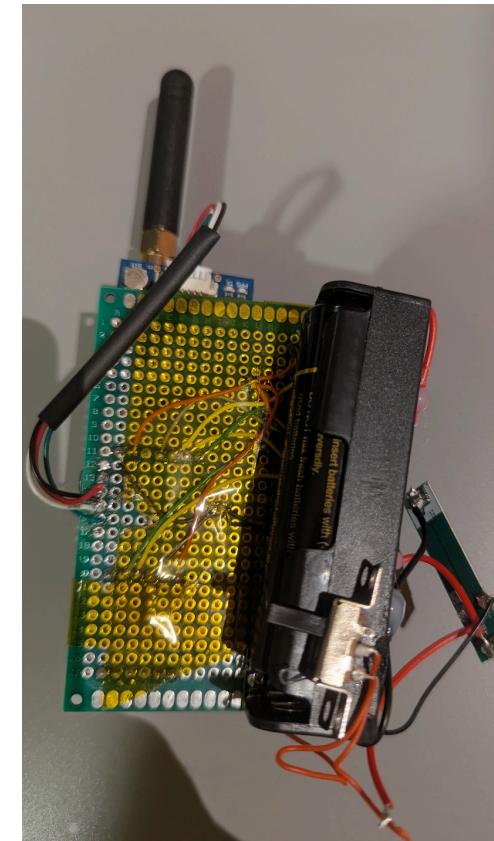
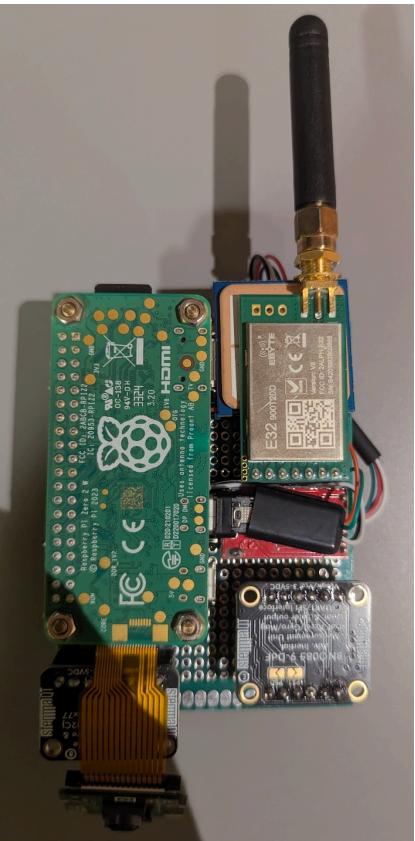
## 4. Selección de tecnologías – Conexión - Raspberry Pi



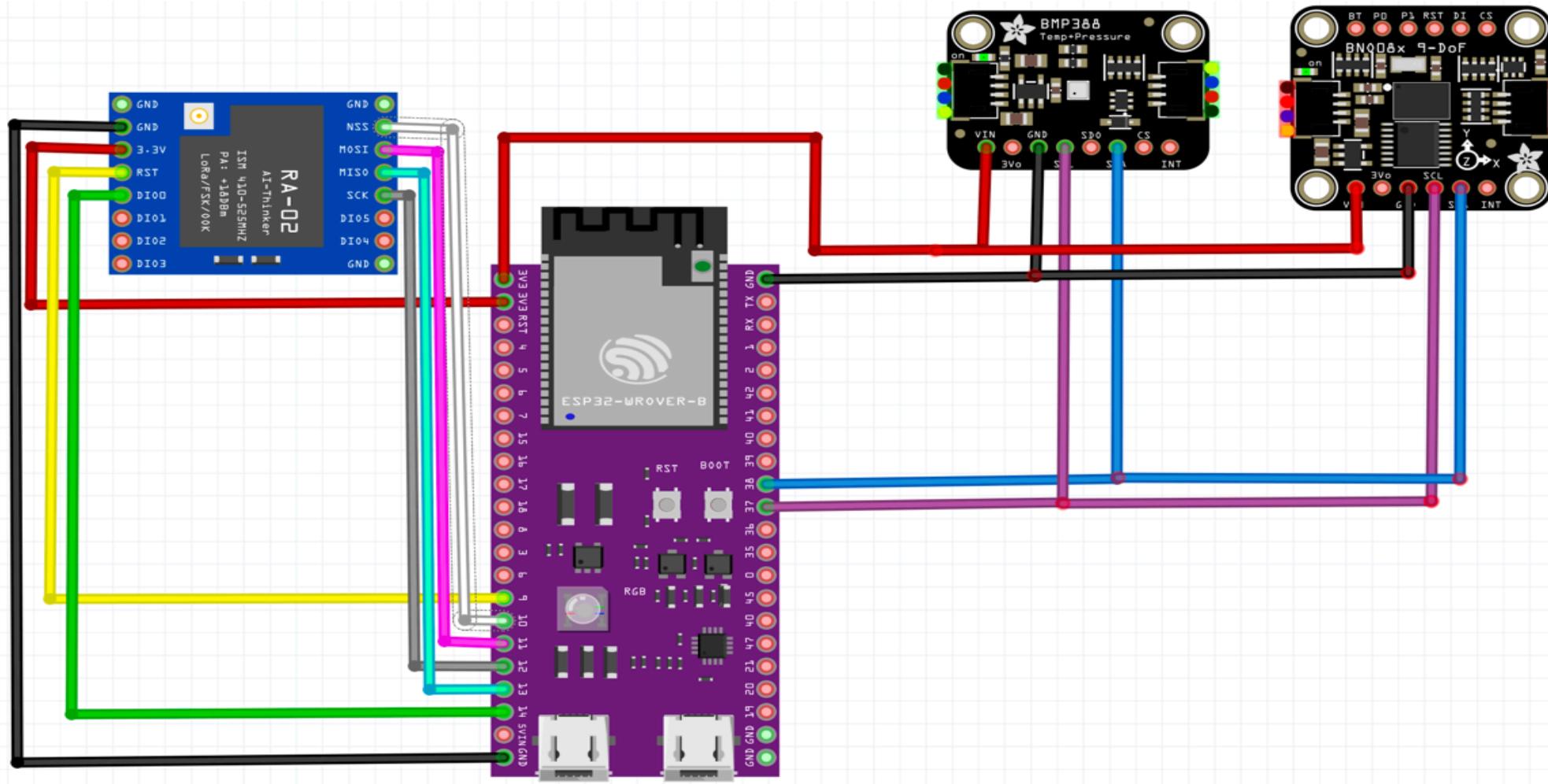
## 4. Selección de tecnologías – Conexión - Raspberry Pi



## 4. Selección de tecnologías – Conexión - Raspberry Pi



## 4. Selección de tecnologías – Conexión - Esp32



## 4. Selección de tecnologías – Interfaz

Lat	Long	GPS Altitude	<a href="#">Download</a>
Unknown	Unknown	Unknown	
<b>Speed</b>	<b>Last Connection</b>	<b>CPU Temp</b>	
Unknown	2025-08-03T04:16:51.642376	55.31	
<b>Altitude</b>	<b>Pressure</b>	<b>Temperature</b>	
0.38	942.75	37.25	
<b>Pitch</b>	<b>Roll</b>	<b>Yaw</b>	
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:**  $\approx 2$  h de operación continua con telemetría + vídeo

- **Persistencia y exportación:**

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

## 6. Conclusiones

- **Objetivo cumplido:** se ha desarrollado una plataforma reutilizable e independiente del hardware para la visualización de datos de un CanSat en tiempo real.
- **Arquitectura modular:** adquisición de datos, transmisión, backend y frontend desacoplados, lo que permite cambios independientes y facilita el mantenimiento y la reutilización.
- **Validación del sistema:** implementación de un CanSat real, demostrando la independencia del hardware y la adaptabilidad a distintos sensores y configuraciones.
- **Gestión del vídeo (Raspberry Pi Zero 2 W):** se detectaron limitaciones de estabilidad, solventadas mediante resolución  $640 \times 480$  a 15 fps y uso de FFmpeg con aceleración por hardware, obteniendo un flujo de vídeo continuo y estable.

## 7. Trabajo futuro

- Incorporar más **gráficas en tiempo real** y visualizaciones personalizadas
- Implementar un **sistema de telemandos** para el control remoto de sensores y parámetros de configuración
- Añadir **mecanismos de autenticación** para proteger el acceso a datos sensibles
- Dar soporte a la monitorización simultánea de **múltiples CanSats**

# Preguntas

CanSat: sistema de adquisición y datos en tiempo real

Sergio García Sánchez

Gracias por su atención