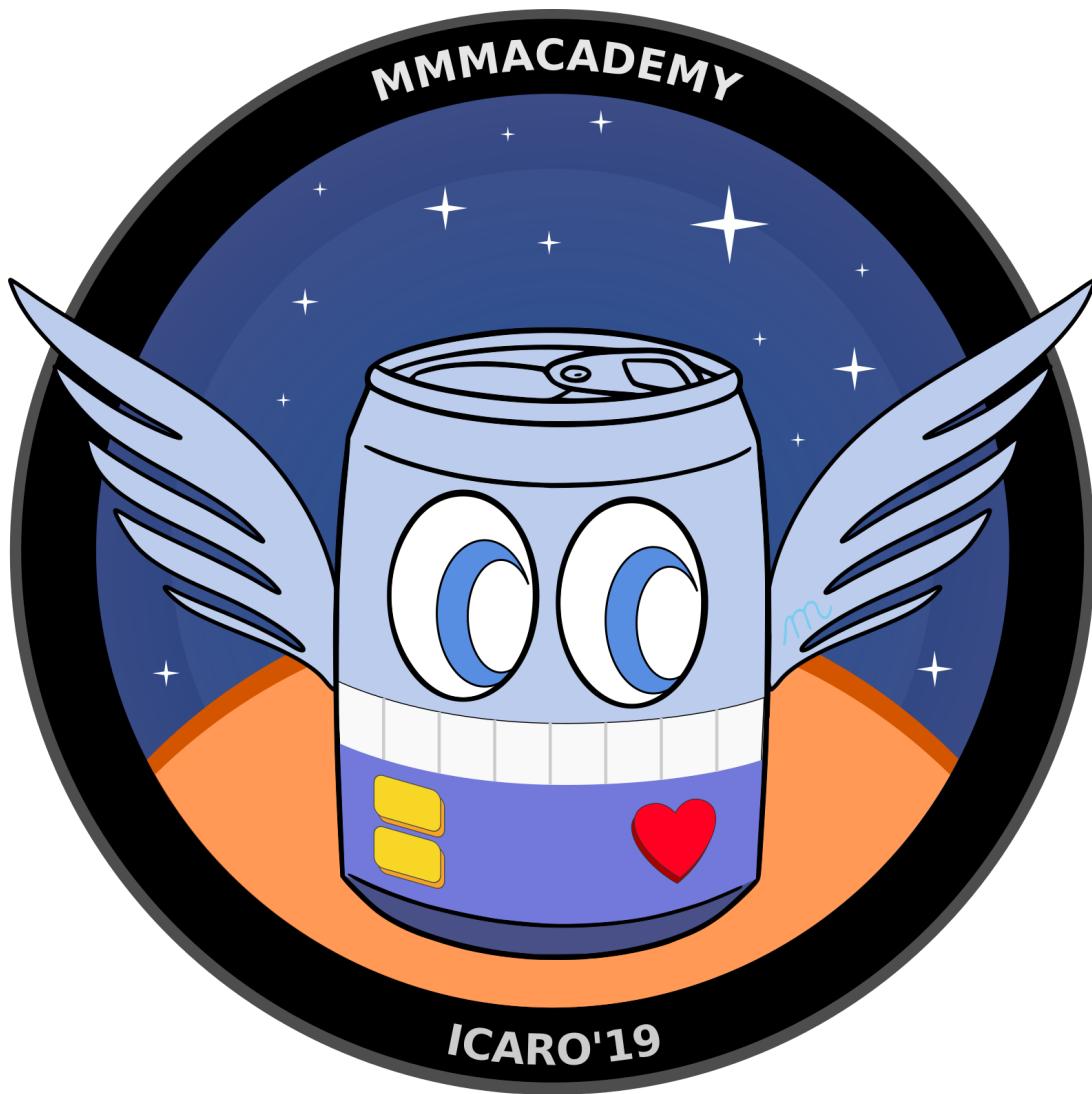


MMM@cademy

# Critical Design Review

Icaro19

---



Mentor: Jose Angel Martínez Domingo

---



|  |           |
|--|-----------|
| <b>1. Introducción y Antecedentes</b>              | <b>3</b>  |
| 1.1 Organización del Equipo y Asignación de Tareas | 4         |
| 1.2 Distribución y Asignación de Tareas            | 5         |
| 1.3 Organigrama                                    | 6         |
| <b>2. Proyecto Cansat</b>                          | <b>7</b>  |
| 2.1 Misión Primaria                                | 7         |
| 2.2 Misión Secundaria                              | 8         |
| 2.3 Diagrama de bloques                            | 12        |
| <b>3. Diseño Mecánico y Estructural</b>            | <b>12</b> |
| 3.1 Diseño Mecánico Cansat                         | 12        |
| 3.2 Diseño mecánico Estación de Tierra             | 16        |
| <b>4. Diseño Eléctrico</b>                         | <b>16</b> |
| 4.1 Esquema Eléctrico                              | 16        |
| 4.2 Alimentación del Cansat                        | 18        |
| 4.3 Telecomunicaciones.                            | 19        |
| 4.3.1 Identificación de paquetes                   | 20        |
| 4.3.2 Antena                                       | 21        |
| <b>5. Programación</b>                             | <b>22</b> |
| 5.1 Lenguaje de programación                       | 22        |
| 5.2 Diagrama de Flujo Cansat                       | 23        |
| 5.3 Diagrama de Flujo Estación de Tierra           | 25        |
| <b>6. Sistema de aterrizaje</b>                    | <b>26</b> |
| <b>7. Presupuesto Detallado</b>                    | <b>27</b> |
| <b>8. Plan de financiación</b>                     | <b>29</b> |
| <b>9. Plan de Actuación</b>                        | <b>30</b> |
| <b>10. Campaña de Difusión</b>                     | <b>31</b> |
| <b>11. Estado del Proyecto</b>                     | <b>32</b> |
| <b>ANEXO 1</b>                                     | <b>33</b> |
| Código Cansat                                      | 33        |
| Introducción                                       | 33        |



---

|                       |           |
|-----------------------|-----------|
| Código                | 33        |
| Código Ground Station | 37        |
| Introducción          | 37        |
| Código                | 38        |
| <b>ANEXO 2</b>        | <b>44</b> |
| PINOUT                | 44        |
| <b>BIBLIOGRAFÍA</b>   | <b>46</b> |



## **1. Introducción y Antecedentes**

El siguiente documento detalla el proyecto realizado por el equipo Icaro19 de la Región de Murcia para su participación en el programa [Cansat](#) de ESERO (European Space Education Resource Office) de acuerdo a sus requerimientos.

Un Cansat es un pequeño satélite del tamaño de una lata de refresco que será elevado hasta una altitud de unos 1000 metros donde empezará su descenso. Durante el descenso el Cansat deberá transmitir los datos de los sensores embarcados a una estación de tierra.



Imagen 1: Ilustración gráfica  
fases del Cansat



## 1.1 Organización del Equipo y Asignación de Tareas

Somos un grupo de estudiantes de Secundaria de diferentes localidades de la Región de Murcia (Alcantarilla, Águilas, Murcia y Era Alta). Todos somos compañeros de clase en la Academia Tecnológica MMMacademy. Aunque habíamos participado anteriormente en ferias y eventos con nuestros proyectos, el programa Cansat nos pareció una gran oportunidad para trabajar por primera vez todos juntos en el mismo proyecto.

Las cualidades que mejor definen a Icaro19 son la capacidad de trabajo en equipo, nuestra ilusión y nivel de compromiso pero sobre todo la pasión que tenemos todos con el mundo tecnológico.

## 1.2 Distribución y Asignación de Tareas

Desde el comienzo repartimos las tareas teniendo en cuenta las fortalezas e intereses individuales de cada miembro del equipo.

Para ello creamos 4 equipos de trabajo:

**-Equipo Tierra:** Responsable de la interfaz de la Estación de Tierra. Su misión es comprobar el correcto funcionamiento de la recepción de las telecomunicaciones así como la programación y desarrollo de la interfaz necesaria para interpretar los datos recibidos y transcribirlos a un modelo gráfico.

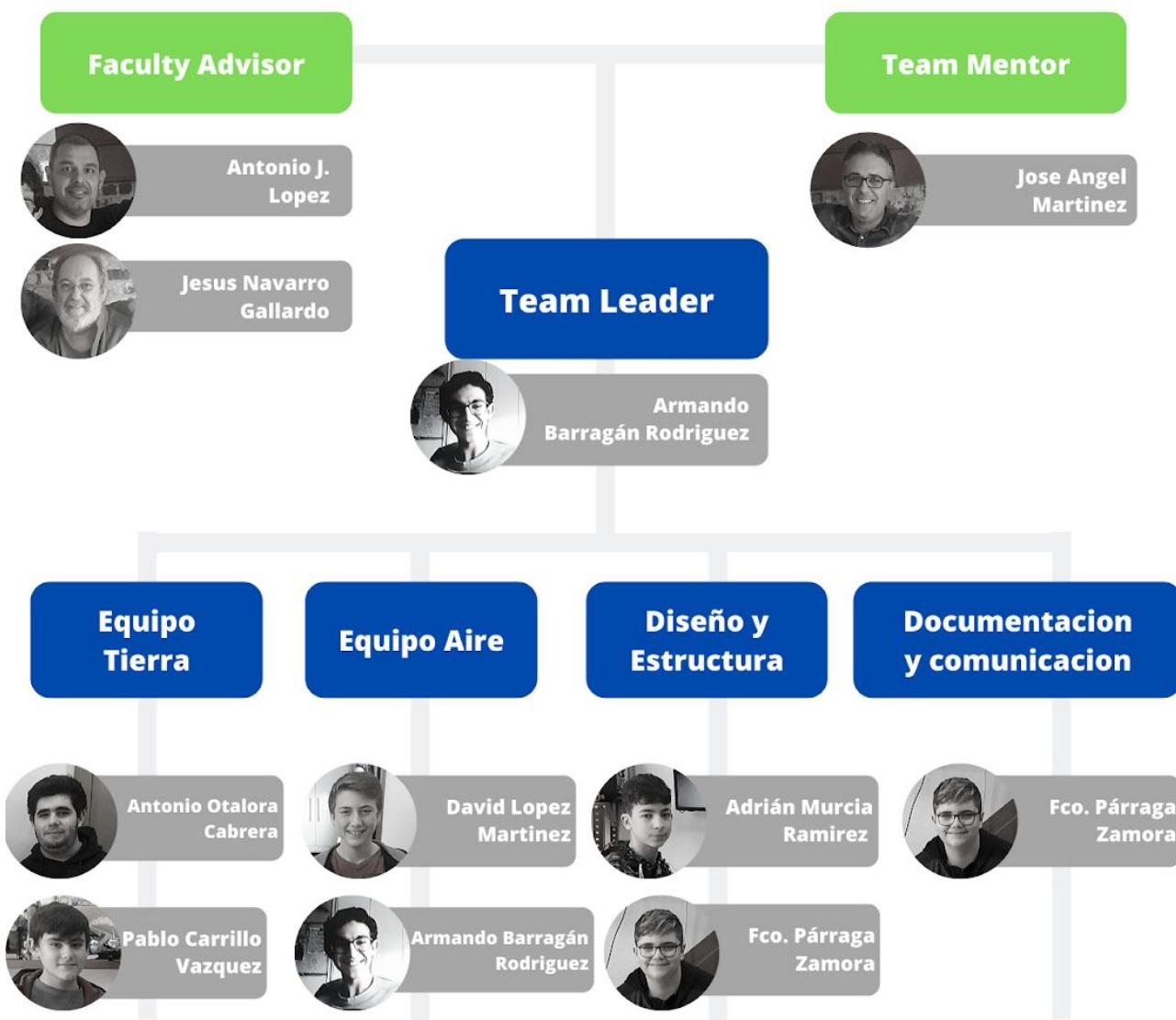
**-Equipo Aire:** Se encargaran de la programación y configuración de los elementos del satélite. También tendrán la responsabilidad de preparar las comunicaciones salientes para que lleguen los datos a la Estación de Tierra y esta a su vez sea capaz de discriminar nuestro mensaje del resto de cansats.

**-Equipo Diseño y Estructura:** Diseño y elaboración de la estructura y encapsulado del Cansat. Serán los encargados de acomodar la electrónica de vuelo, sensores y equipamiento del Cansat.



**-Documentación y Comunicación:** En este apartado deberán documentar todos los avances, mantener actualizado el repositorio en Github, elaborar y ejecutar la estrategia de comunicación y organizar y moderar las reuniones del equipo.

### 1.3 Organigrama





## **2. Proyecto Cansat**

### **2.1 Misión Primaria**

El objetivo de la misión primaria es medir temperatura y presión durante el descenso del Cansat y además transmitir los datos a la Estación de Tierra durante el viaje.

Dado que hemos elegido Raspberry Pi Zero como Unidad de Procesamiento Central (Microprocesador) y Python como entorno de programación debíamos buscar sensores con librerías suficientes y que ejecutarán las misiones de la manera más eficaz posible. Así para esta misión hemos elegido el BMP280.

Hemos realizado la comunicación por puerto I2C, esto nos garantiza reducir y simplificar el cableado. Para ello cada sensor debe transmitir por direcciones distintas.



Imagen 2: Sensor BMP280

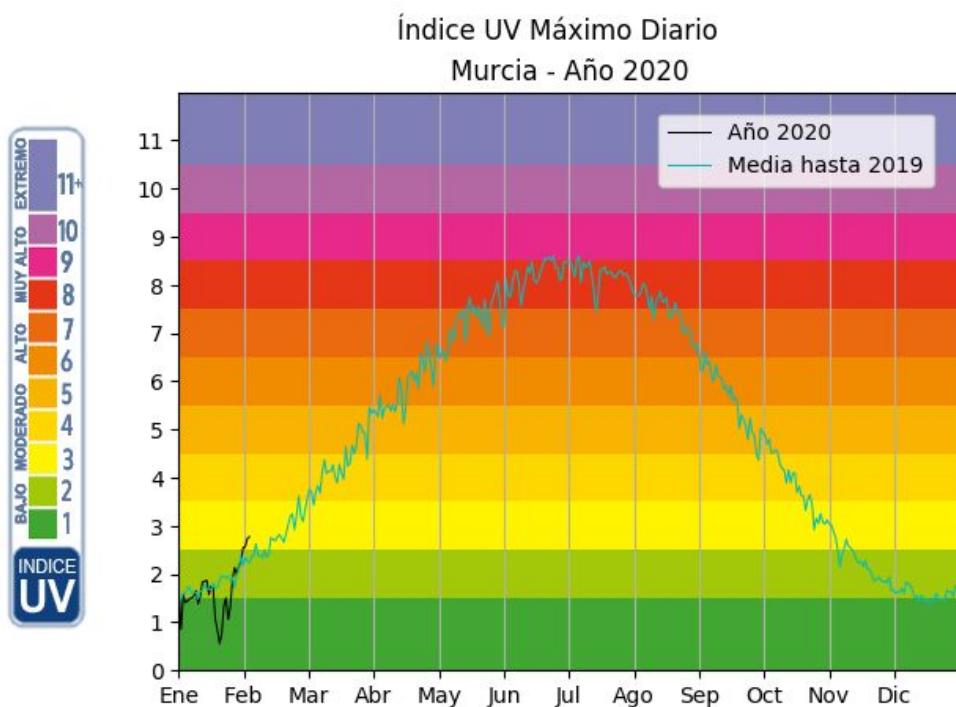
Especificaciones:

- Bosch BMP280 (Temperatura, presión y altitud)
- 3,3V o 5V
- Interfaz I2C
- Protección de Polaridad inversa
- Dimensiones: 19x19x2.75mm (LxWxH)
- Compatible pines Raspberry
- Margen de error: 1hPa, 1°C, 1m.



## 2.2 Misión Secundaria

Somos un grupo de estudiantes muy concienciados con el medio ambiente y por esto, queríamos que nuestra misión secundaria tuviera que ver directamente con ello. Vivimos en la Región de Murcia, donde año tras año vemos como aumentan las temperaturas y como asciende el índice de radiación solar.



© Agencia Estatal de Meteorología

AEMet  
Agencia Estatal de Meteorología

Imagen 3: Registro radiación UV provincia de Murcia 2019 y primeras lecturas 2020 Agencia Estatal de Meteorología

Por ello decidimos aprovechar la oportunidad que nos brindaba el proyecto Cansat de poder hacer mediciones de radiación UV a diferentes alturas y estudiar la relación directa con la cantidad de CO<sub>2</sub> y las medidas de TVOC (Compuestos orgánicos volátiles).



Para esta misión elegimos los siguientes elementos:

### SGP30 Sensor de Calidad del Aire



Imagen 4: Sensor SGP30

Especificaciones:

- Sensiron SGP30 TVOC y eCO2
- TVOC sensing from 0-60,000 ppb (parts per billion)
- CO2 sensing from 400 to 60,000 ppm (parts per million)
- Interfaz I2C
- 3,3v - 5v
- Protección de polaridad inversa
- Medidas: 19X19X3

### VEML6075 UVA/B

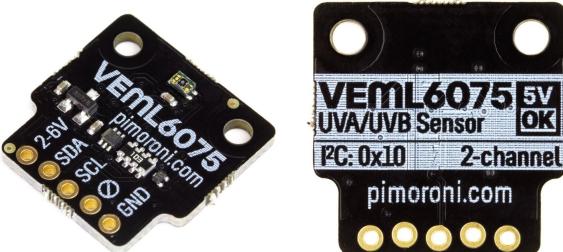


Imagen 5: Sensor VEML6075



## Especificaciones:

- sensor VEML6075 UVA/B
- Detecta índices UVA / UVB y UVA / UVB sin procesar e índices promedio
- 3.3V - 5V
- Interfaz I2C
- Protección de polaridad inversa
- medidas: 19X19X3mm

Para poder darle más precisión a los datos, hemos optado por integrar en nuestra misión secundaria un sensor GPS con el que cruzar las lecturas con datos de posición. Hemos elegido para este fin:

## **Adafruit Ultimate GPS**

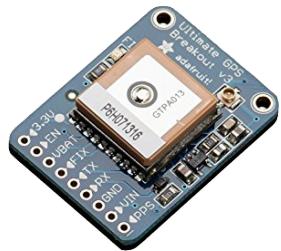


Imagen 6: Ultimate GPS V3

## Especificaciones:

- Sensibilidad de -165 dBm, actualizaciones de 10 Hz, 66 canales
- 5V - 20mA
- batería RTC
- Registro de datos Incorporado
- Salida PPS
- Antena cerámica incorporada
- LED indicador de posicionamiento (FIX)
- medidas: 25.5 X 35 X 6.5 mm



Dado que vamos a desarrollar una interfaz gráfica propia en la Estación de Tierra, hemos querido dar datos de orientación y posición, por ello hemos elegido un sensor de movimiento acelerómetro que nos ayude a representar la posición de nuestro Cansat durante el descenso. *Hemos optado por sustituir el módulo GPS inicial de Keyestudio por el Ultimate GPS de Adafruit ya que integraba mejor en nuestro sistema y optimizamos espacio y rendimientos de consumo.*

### LSM303D 6DoF Motion Sensor



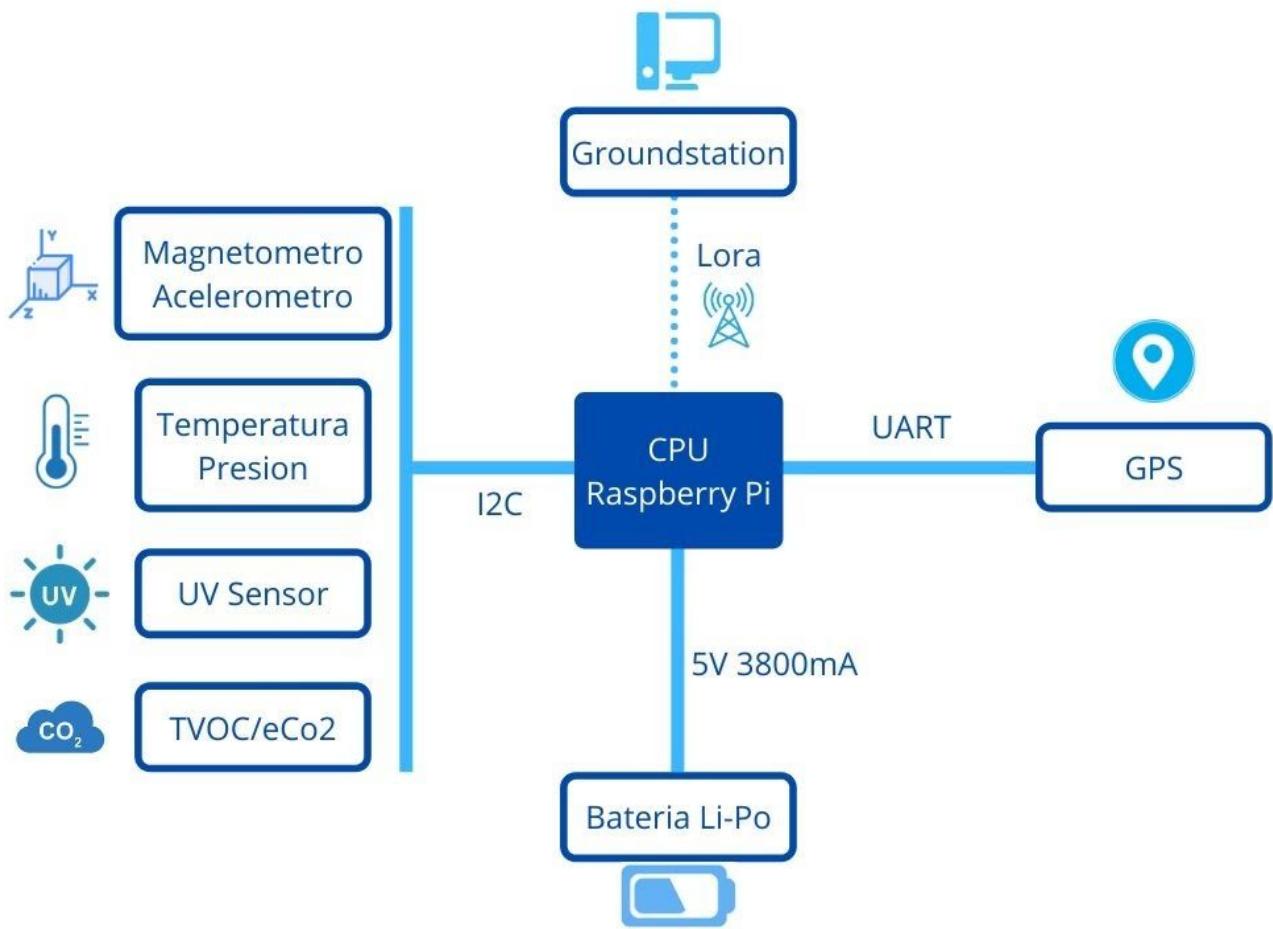
Imagen 7: Sensor LSM303D

#### Especificaciones:

- LSM303D 6DoF Motion Sensor
- $\pm 2/\pm 4/\pm 8/\pm 12$  gauss magnetic scale
- $\pm 2/\pm 4/\pm 6/\pm 8/\pm 16$  g linear acceleration
- 16 bit data output
- 3.3V - 5V
- Interfaz I2C
- Protección de polaridad inversa
- medidas: 19X19X3mm



## 2.3 Diagrama de bloques



## 3. Diseño Mecánico y Estructural

### 3.1 Diseño Mecánico Cansat

Las normas de la competición exigen que el Cansat tenga unas medidas de no más de 115 mm de alto y 65 mm de diámetro.



Como disponemos de impresoras 3D, nos pareció ideal para aplicarla a nuestro diseño, primero porque reducía enormemente los costes de fabricación de los prototipos y en segundo lugar porque nos facilita la posibilidad de modificar las versiones del diseño tantas veces como necesitemos.

Nuestro diseño estructural se divide en dos partes:

**Estructura Interior:** Debíamos crear una estructura que fuese capaz de albergar toda la electrónica, módulos, sensores y baterías. Hemos planificado un esqueleto interior de dos pisos. *Hemos variado la posición inicial de la CPU orientando puertos y conexiones hacia el exterior lo que facilita el acceso y la programación.*

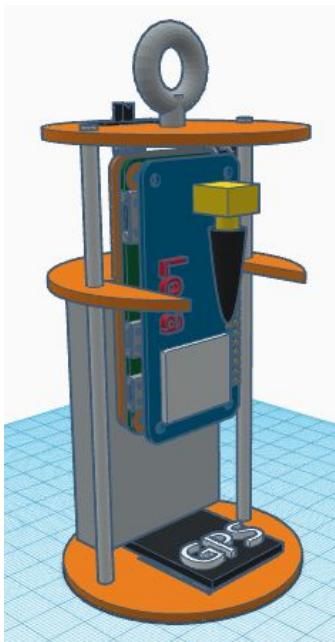


Imagen 8: Primera Versión Estructura

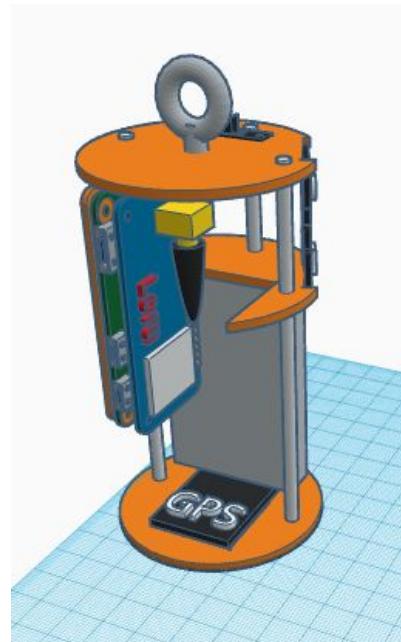


Imagen 9: Versión final Estructura



Imagen 10: Vista Frontal  
Estructura



Imagen 11: Vista trasera  
Estructura



El compartimento superior lo hemos destinado a la parte de sensores. Y la parte inferior hemos pensado que sería más adecuada para los elementos más pesados como la batería y el GPS ya este peso podría ayudar a facilitar el vuelo y la posición correcta del Cansat en el descenso actuando como un lastre.

**Diseño Exterior:** Para el exterior hemos realizado un encapsulado de unos 2mm de grosor. Aunque no hemos hecho todavía pruebas de impacto pensamos que será suficiente para proteger nuestro diseño. *Sobre el diseño exterior se ha realizado una modificación consistente en añadir una guia/nervio interior que ayuda a que el esqueleto interior permanezca más fijo y no gire sobre sí mismo.*

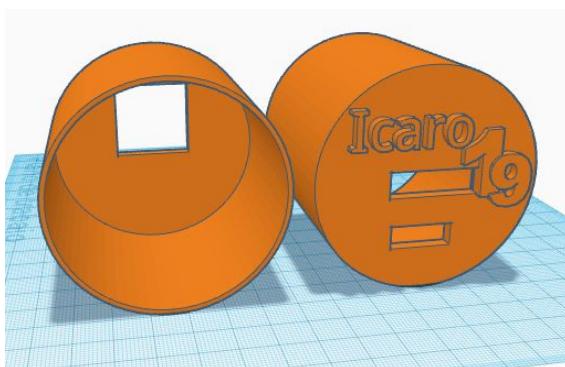


Imagen 12: Versión inicial de estructura exterior

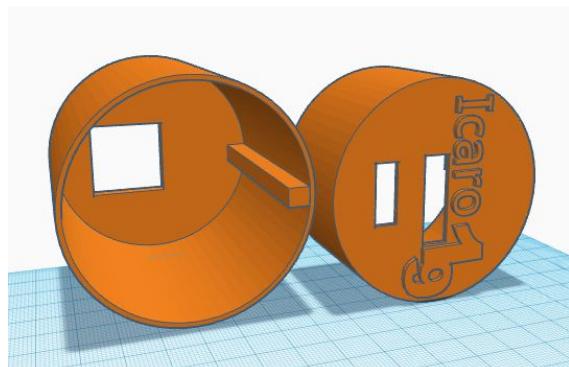


Imagen 13: Versión final de estructura exterior



Imagen 14: Detalle de encastre en nervio/guia interior. Tornilleria M3



### 3.2 Diseño mecánico Estación de Tierra

Hemos programado nuestra aplicación con PyQt para poder ejecutar desde un ordenador portátil. Ya que vamos a recibir los datos de nuestro Cansat a través de la red Lora, utilizaremos un módulo [SX1262 de Waveshare](#) en modo escucha para recibir datos a través del serial.

Para incrementar el alcance de la recepción hemos optado por construir nuestra antena Yagui que conectaremos al módulo Lora y este a su vez por USB al PC.

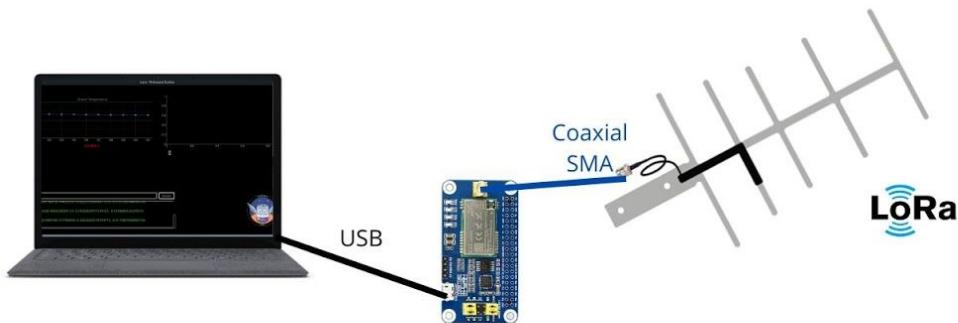


Imagen 15:  
Elementos de la  
Estación de tierra

## 4. Diseño Eléctrico

### 4.1 Esquema Eléctrico

El diseño eléctrico lo hemos planteado intentando meter la mayor parte de sensores en el bus I2C. Con esto hemos conseguido optimizar al máximo el cableado y ahorrar espacio. La dificultad en este sentido es asegurar que cada sensor emite por una dirección distinta. Adjuntamos la tabla de direccionamiento del BUS.



| Sensor   | Bus | Dirección |
|----------|-----|-----------|
| BMP280   | I2C | 0X76      |
| SGP30    | I2C | 0X58      |
| VEML6075 | I2C | 0X10      |
| LSM303D  | I2C | 0X1D      |

Según requerimientos de la organización hemos incluido un microinterruptor de conmutador doble accesible desde el exterior del cansat para facilitar el encendido y apagado.

Hemos creado un sistema modular. El cuerpo principal lo conforman el procesador principal (Raspberry Pi Zero) y el módulo LoRa. Hemos creado dos concentradores para los sensores en lugar de integrarlo todo en la misma PCB, esto nos facilita mucho la tarea de acomodar los componentes y supone una ventaja a la hora de localizar averías y sustituir elementos averiados.

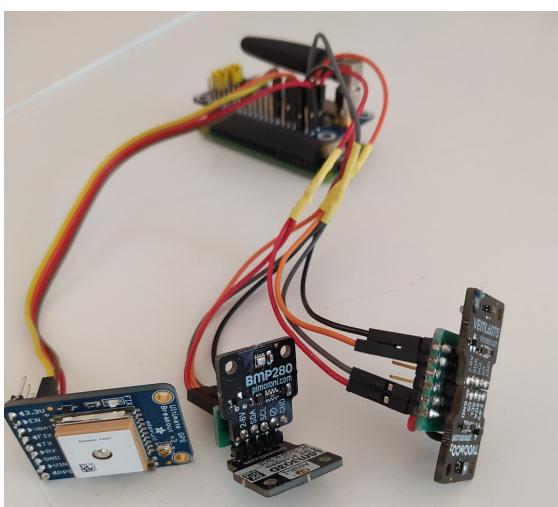


Imagen 16: Unidad Central CPU más módulos de sensores y GPS

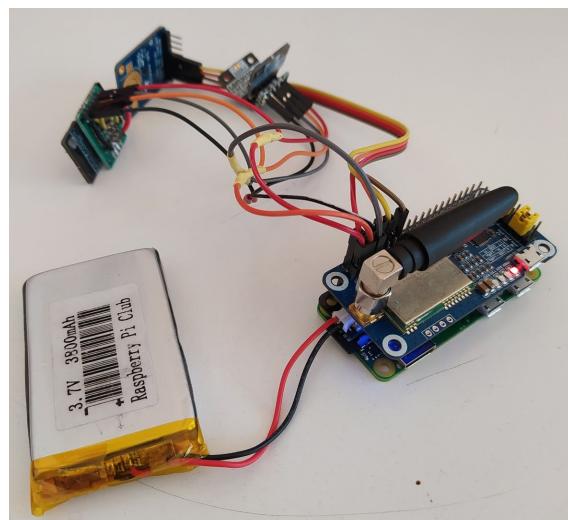


Imagen 17: Sistema completo más batería

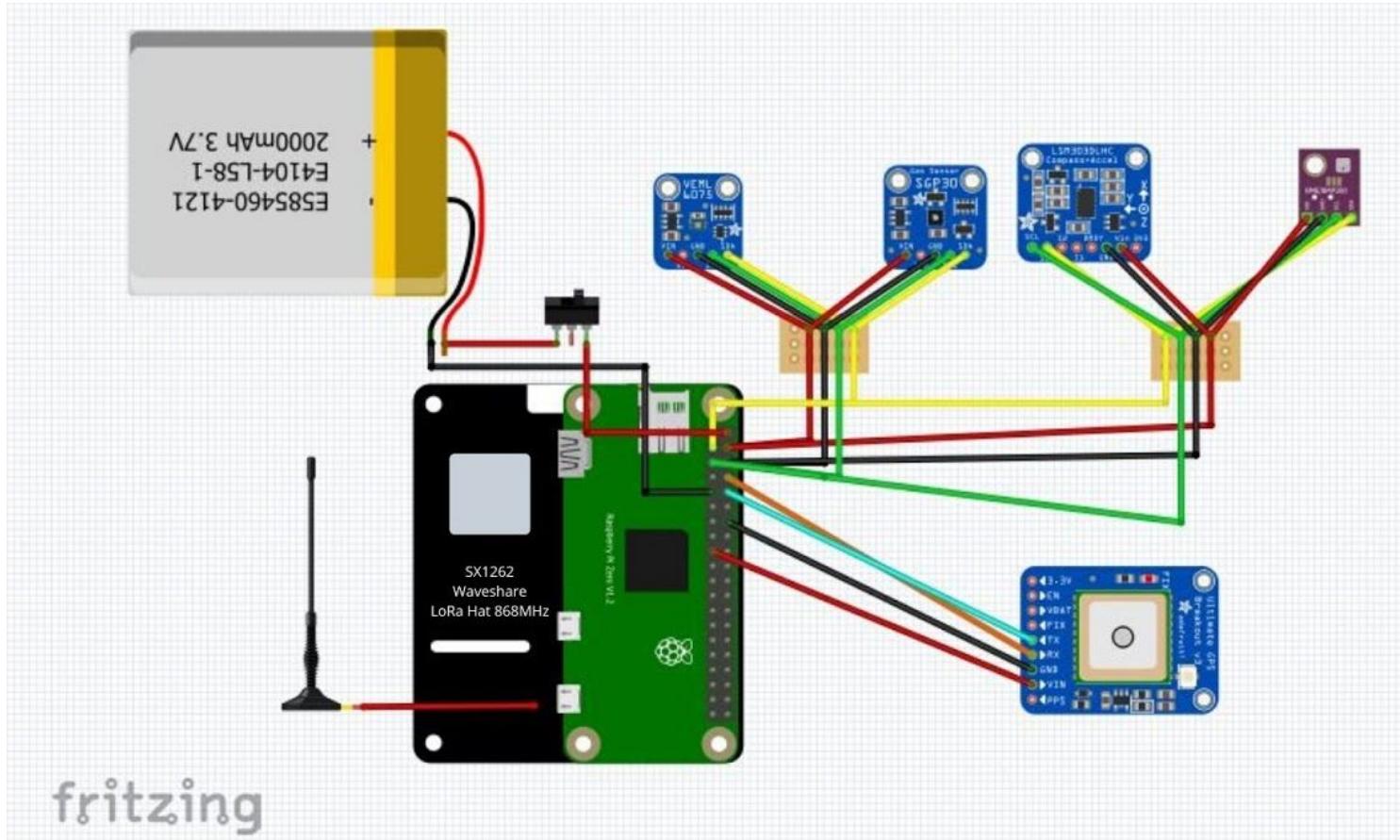


Imagen 18: Esquema eléctrico Cansat

## 4.2 Alimentación del Cansat

Para la alimentación del Cansat hemos tenido en cuenta el consumo de todos los componentes. Hemos integrado en nuestra Raspberry Pi un [modulo LiPo SHIM](#) con el que garantizamos un voltaje continuo de 5V y 1,5 amperios a nuestra Raspberry y el resto del sistema. Este módulo es compatible con el conector JST de la batería Li-Po de nuestro sistema.

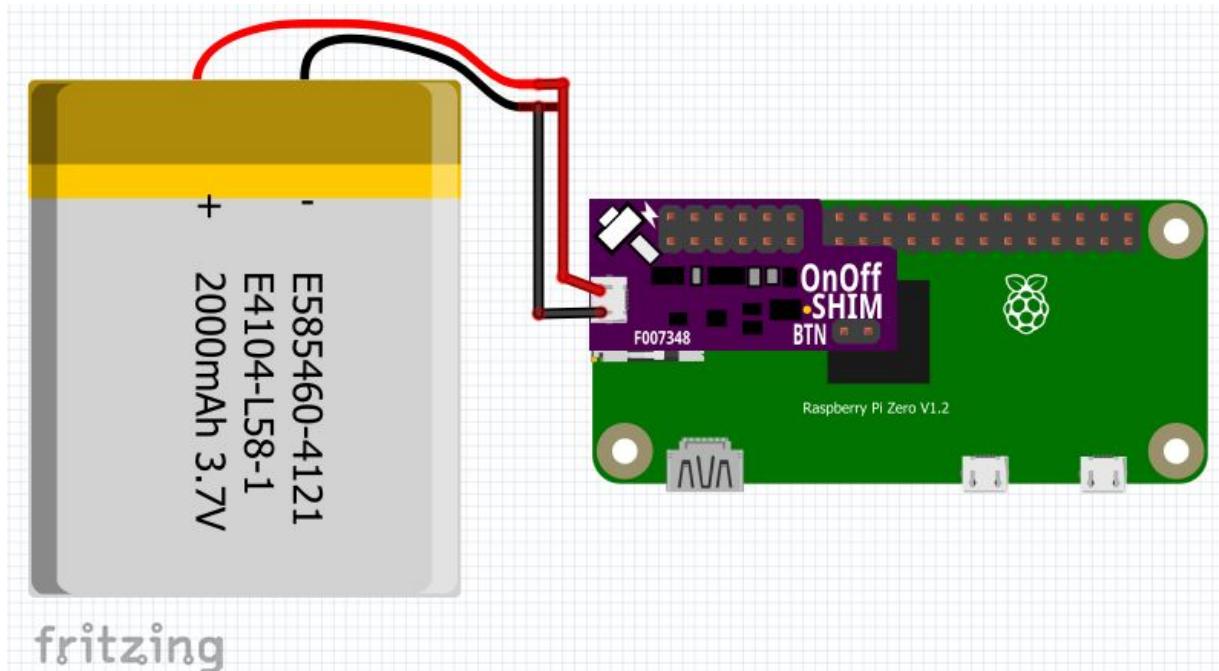


Imagen 19: Esquema alimentación Cansat

#### PINOUT del Sistema Completo en ANEXO 2

### 4.3 Telecomunicaciones.

Para nuestro proyecto estuvimos barajando varias posibilidades teniendo en cuenta el alcance y el ancho de banda. Además, debía ser una banda no licenciada. Tuvimos que decidir entre Sigfox o LoRa.



|                                 | SIGFOX            | LORA                      |
|---------------------------------|-------------------|---------------------------|
| <b>Frequency band</b>           | 868/902 MHz (ISM) | 433/868/780/915 MHz (ISM) |
| <b>Urban range</b>              | 3-10km            | 2-5km                     |
| <b>Rural range</b>              | 30-50km           | 15-20km                   |
| <b>Packet size</b>              | 12 bytes          | Defined by user           |
| <b>Devices per access point</b> | 1M                | 100k                      |
| <b>Status</b>                   | In deployment     | Spec released June 2015   |
| <b>Topology</b>                 | Star              | Star                      |

Imagen 20: Comparativa Sigfox Vs Lora

Al final escogimos LoRa en la banda de 868MHz por las siguientes características:

- Alta tolerancia a las interferencias
- Alta sensibilidad para recibir datos (-168dB)
- Basado en modulación “chirp”
- Bajo Consumo
- Largo alcance 10 a 20km
- Permite conexiones punto a punto

También nos gusto porque la modulación “Chirp” se usa desde hace décadas en las comunicaciones espaciales y nos parecía estupendo para nuestra “carrera espacial” particular.

#### 4.3.1 Identificación de paquetes

Para evitar interferencias con otros equipos que usen la misma tecnología hemos decidido incluir un código inicial en todos nuestros envíos de información en el cansat a la Estación de Tierra. Así el receptor en la Estación de Tierra podrá discriminar los paquetes o interferencias recibidas sin el código inicial correcto y solo almacenará y graficara los envíos correctos.



### 4.3.2 Antena

Después de las pruebas de transmisión que realizamos, descubrimos que la antena integrada en el módulo de la Estación de Tierra tenía un alcance limitado a unos 500 o 600 metros así que decidimos fabricar nuestra antena Yagui.

Para los cálculos de longitud y posición de los elementos utilizamos la siguiente web:

<https://www.rfwireless-world.com/calculators/3-element-Yagi-Antenna-Calculator.html>

En ella puedes obtener los cálculos introduciendo la banda en la que quieras trabajar.

Para la fabricación reciclamos un tendedero de varillas.



Imagen 21: Tendedero antes del reciclaje



Imagen 22: Antena de tres elementos



## **5. Programación**

### **5.1 Lenguaje de programación**

El desarrollo de las aplicaciones se está realizando entre dos equipos: Equipo Tierra (ET) y Equipo Aire (EA). El lenguaje utilizado para el desarrollo de las aplicaciones es **Python**, ya que es un lenguaje ampliamente utilizado, de código abierto, amigable y multiplataforma. Todos los componentes que hemos elegido tienen sus librerías en Python, por lo que nos facilita su utilización. Al ser multiplataforma podemos programar tanto el cansat como la estación de tierra utilizando el mismo lenguaje. Uno de los puntos fuertes de Python es la amplia comunidad de desarrolladores que giran en torno a él y buscan aportar, compartir y construir software escalable en comunidad, esto nos facilita la búsqueda de recursos y la resolución de problemas durante el desarrollo.

Uno de los retos a los que nos enfrentamos es desarrollar una estación de tierra para mostrar los datos recogidos en tiempo real y graficar estos valores. Tras buscar información sobre las distintas posibilidades, decidimos que éstas debían cumplir tres requisitos imprescindibles:

**1- Total integración con Python**

**2- Que disponga de bastante documentación**

**3- Que tenga herramientas que faciliten el diseño.**

Considerando estos puntos encontramos dos, PyQt5 y PyGTK, ambas tienen un editor visual que hacen que el diseño de la interfaz sea fácil y rápido. La elegida



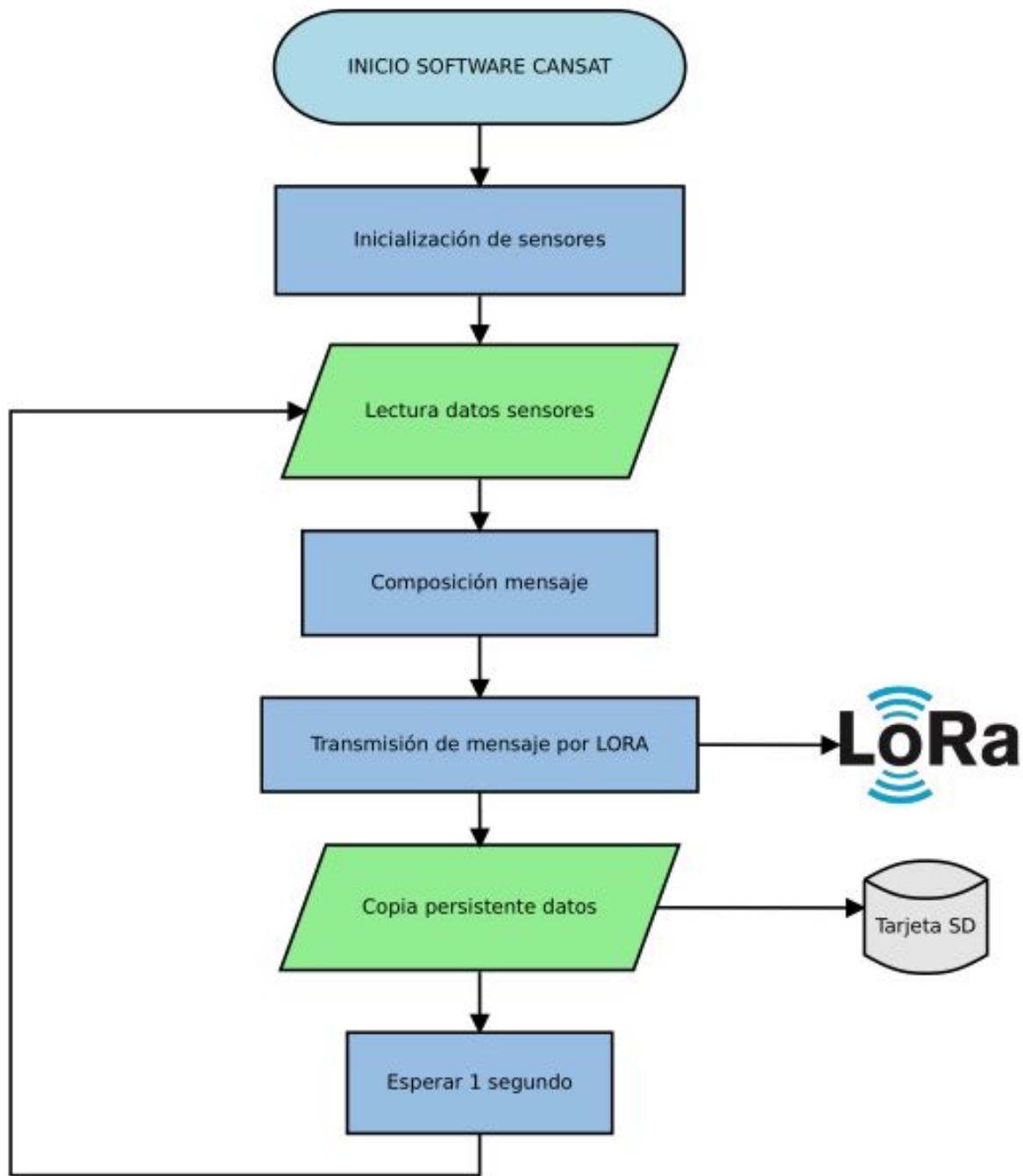
---

para el desarrollo de la estación de tierra es **PyQt5** porque consideramos que su documentación es más clara, existen más recursos y es ampliamente utilizada en cualquier sistema.

Antes de comenzar con el desarrollo de las aplicaciones hemos recibido una formación básica en programación con Python y PyQt5, y tenemos que decir que nos encanta éste lenguaje y es fácil de programar, aunque la parte más complicada la encontramos en la interfaz gráfica, ya que requiere de unos conocimientos algo más avanzados como es la programación orientada a objetos.

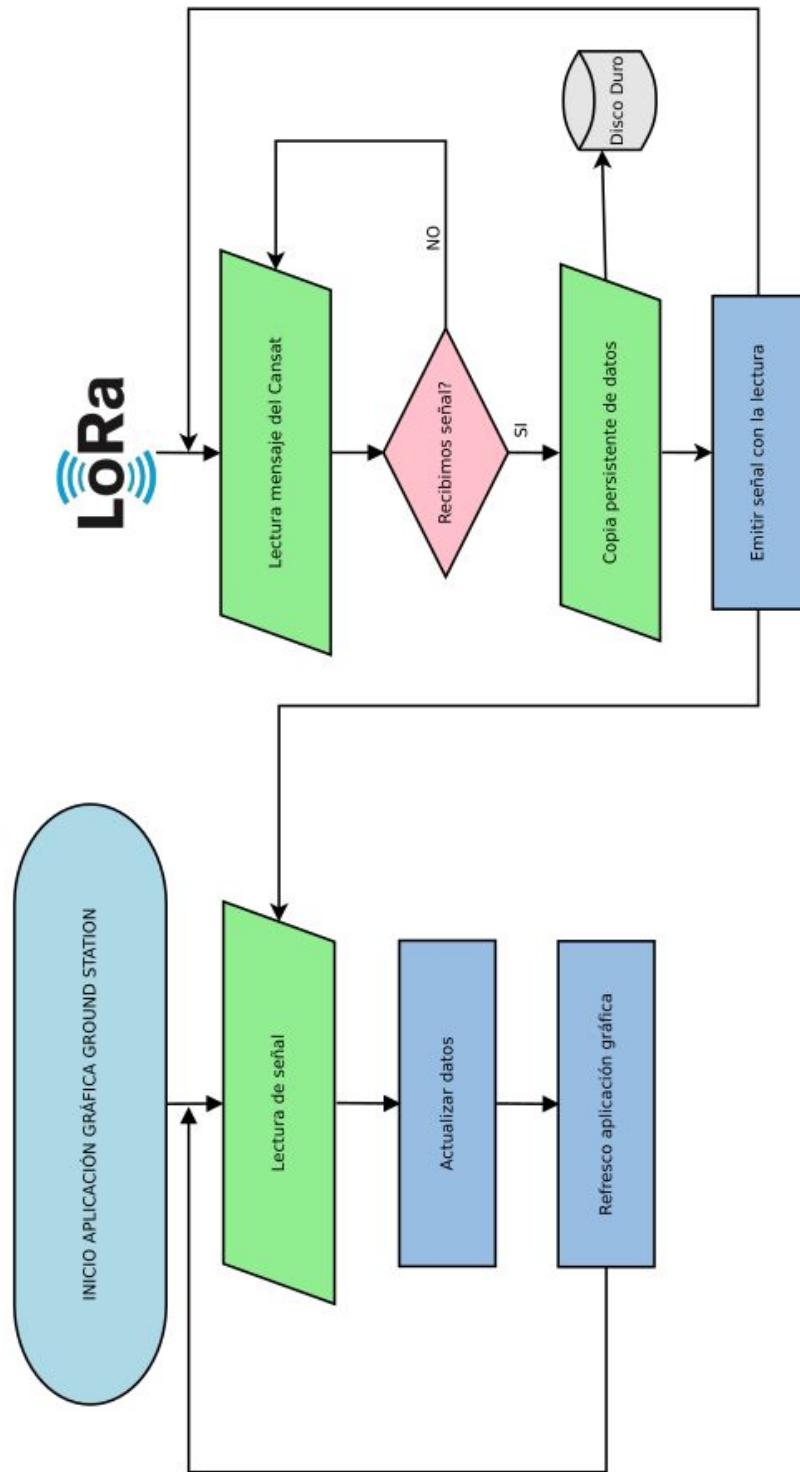
**Adjuntamos código de Estación de Tierra y Código de Cansat comentado y explicado en ANEXO 1**

## 5.2 Diagrama de Flujo Cansat





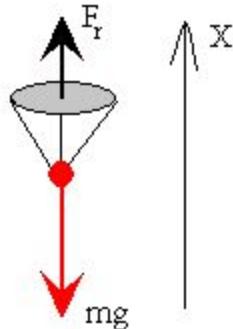
### 5.3 Diagrama de Flujo Estación de Tierra





## 6. Sistema de aterrizaje

Para la construcción de nuestro paracaídas contamos con la ayuda del equipo de plegadores civiles y militares de la Base Aérea de Alcantarilla. Ellos nos enseñaron la física implicada en el proceso de descenso y el cálculo del radio del mismo.



Los requerimientos de la prueba indican que la masa de nuestro Cansat debe estar entre 300 y 350gr, que el tiempo de vuelo no debe ser mayor de 120 segundos. También indican que la velocidad de descenso recomendada debe estar entre 8 y 11 m/sec. Con estos datos y teniendo en cuenta que nuestro paracaídas es de geometría circular el coeficiente de resistencia ( $cd$ ) será de 1.2 así que realizados los cálculos estimamos que nuestro paracaídas debía tener un radio de entre 60 a 65 cm.

Además del diámetro del paracaídas, hemos tenido en cuenta algunas indicaciones del personal de la Base Aérea donde nos recomendaban un sistema de redecilla en lugar de las típicas cuerdas denominadas líneas. También nos aconsejaron que la distancia de la campana hasta nuestro Cansat debe ser unos 10cm más que el largo de la campana y la red.



Imagen 23: Paracaídas Terminado, malla y campana



## **7. Presupuesto Detallado**

Componentes:

Raspberry Pi Zero

<https://shop.pimoroni.com/products/raspberry-pi-zero-w>

LiPo Shim

<https://shop.pimoroni.com/products/lipo-shim>

LSM303D 6DoF Motion Sensor Breakout

<https://shop.pimoroni.com/products/lsm303d-6dof-motion-sensor-breakout>

BME280 Breakout - Temperature, Pressure, Humidity Sensor

<https://shop.pimoroni.com/products/bmp280-breakout-temperature-pressure-altitude-sensor>

SGP30 Air Quality Sensor Breakout

<https://shop.pimoroni.com/products/sgp30-air-quality-sensor-breakout>

VEML6075 UVA/B Sensor Breakout

<https://shop.pimoroni.com/products/veml6075-uva-b-sensor-breakout>

Adafruit Ultimate GPS V3

[https://tienda.bricogeek.com/modulos-gps/442-adafruit-gps-mtk3339.html?search\\_query=gps&results=17](https://tienda.bricogeek.com/modulos-gps/442-adafruit-gps-mtk3339.html?search_query=gps&results=17)

103450 3.7V 1800Mah Batería Recargable de Litio

[https://www.amazon.es/TOOGOO-Recargable-Pol%C3%ADmero-Grabadora-Auriculares/dp/B07M8ZHW2Z/ref=sr\\_1\\_12?\\_\\_mk\\_es\\_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=1PRD6XQRYXRQ6&keywords=lipo+2000mah&qid=1580904756&sprefix=lipo+2000%2Caps%2C166&sr=8-12](https://www.amazon.es/TOOGOO-Recargable-Pol%C3%ADmero-Grabadora-Auriculares/dp/B07M8ZHW2Z/ref=sr_1_12?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=1PRD6XQRYXRQ6&keywords=lipo+2000mah&qid=1580904756&sprefix=lipo+2000%2Caps%2C166&sr=8-12)

Waveshare SX1262 Lora Hat

[https://www.amazon.es/Waveshare-Multi-Level-Configuration-Fixed-Point-Transmission/dp/B07VQZ5ZXY/ref=sr\\_1\\_2?\\_\\_mk\\_es\\_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=lora+waveshare&qid=1580905407&sr=8-2](https://www.amazon.es/Waveshare-Multi-Level-Configuration-Fixed-Point-Transmission/dp/B07VQZ5ZXY/ref=sr_1_2?__mk_es_ES=%C3%85M%C3%85%C5%BD%C3%95%C3%91&keywords=lora+waveshare&qid=1580905407&sr=8-2)



| Nombre              | IMG | Cantidad | Precio Unitario | Total  |
|---------------------|-----|----------|-----------------|--------|
| Raspberry Pi Zero W |     | 1,00     | 10.97€          | 10.97€ |
| LiPo Shim           |     | 1,00     | 12,70 €         | 12.70€ |
| LSM303D             |     | 1,00     | 10,97 €         | 10.97€ |
| BMP280              |     | 1,00     | 9,20 €          | 9.20€  |
| SGP30               |     | 1,00     | 19,47 €         | 19.47€ |
| VEML6075            |     | 1,00     | 8,50 €          | 8.5€   |
| GPS                 |     | 1,00     | 24,99 €         | 24.99€ |



|                           |  |      |         |                |
|---------------------------|--|------|---------|----------------|
| LiPo Bateria              |  | 1,00 | 7,08 €  | 7.08€          |
| Waveshare SX1262 Lora Hat |  | 2,00 | 35,99 € | 71.98€         |
| TOTAL                     |  |      |         | <b>185.81€</b> |

## 8. Plan de financiación

Es el primer proyecto que acometemos de esta envergadura, hemos contado con el apoyo y financiación de MMMacademy, nuestra academia, que nos ha aportado todo el equipamiento necesario y nos han permitido utilizar las herramientas y recursos que tienen, además siempre son los primeros en apoyarnos en nuestras iniciativas.

Por supuesto a nuestras familias que nos llevan y traen y también aportan al desarrollo del proyecto.

Hemos iniciado contactos con dos empresas tecnológicas locales a las que les interesa nuestro proyecto y que se muestran receptivas a patrocinar nuestra aventura.



## **9. Plan de Actuación**

Hemos organizado nuestro plan de actuación de la siguiente forma:

- **Llegada y despliegue:** Hemos elaborado un “Checklist” de arranque con el que ir punteando la comprobación y correcto funcionamiento de todos los sistemas. Esta fase contempla la puesta en marcha de la Estación de Tierra y las pruebas de comunicaciones. Una vez superado el Checklist el Cansat estará listo para entregar al personal de la organización.
- **Prelanzamiento:** Se realizarán las comprobaciones de peso, altura y diámetro para que el Cansat se ajuste al reglamento, ante cualquier imprevisto se rectifica en el acto. Justo antes del lanzamiento se realizará un “Fast-Check” con la emision y recepcion de un paquete de datos de prueba para comprobar que las comunicaciones estén activas.
- **Lanzamiento:** Durante esta Fase, nuestro equipo de tierra estará operando nuestra Estación de tierra en espera de señal mientras que el equipo de estructura y diseño desplegaran la antena y apuntan al cohete para hacer seguimiento.
- **Apogeo y Descenso:** Llegados a esta fase, el cohete se abrirá y dejará caer los Cansat. Nuestro equipo de tierra registrará los datos en tiempo real recibiendo una lectura cada 1 o 2 segundos. Además de graficarlos en la aplicación, quedará un registro en formato .CSV para poder tratarlos a posterior.
- **Recuperación:** Durante esta fase nuestro equipo de aire se dispondrá a recuperar el Cansat en base a las últimas coordenadas recibidas por GPS. Se comprobará el estado del Cansat y se recoge la MicroSD que contendrá otra copia de los datos enviados.
- **Análisis y Conclusiones:** Una vez recuperada la MicroSD del Cansat y volcados los datos de la Estación de Tierra, se cruzaran para descartar errores y duplicidades, se analizaran y se entregaran los informes de la misión primaria y secundaria a la organización.



## **10. Campaña de Difusión**

Hemos creado una campaña de difusión orientada a Redes Sociales, para ello hemos creado cuentas en Facebook, Instagram y Twitter con las que pretendemos dar a conocer nuestros avances y progreso en el proyecto y hacer llegar la información de la competición a más personas interesadas. También estamos usando estas cuentas para seguir al resto de equipos participantes y poder estrechar lazos y crear comunidad.

Estamos creando un repositorio en Github que no solo nos sirva a nosotros para documentar nuestro proyecto sino que además, sirva de fuente para futuros participantes.

Hemos pensado en llevar el proyecto a la próxima Feria Maker y contarles nuestra experiencia a los asistentes. Para ello vamos a crear una web con toda la información a modo de diario.



[@icaro19murcia](https://twitter.com/icaro19murcia)



[icarodiecinueve.murcia](https://www.facebook.com/icarodiecinueve.murcia)



[@icaro19\\_mur](https://www.instagram.com/@icaro19_mur)



**GitHub**



## 11. Estado del Proyecto

Actualmente el proyecto se encuentra en fase de pruebas definitivas. En la Estación de tierra solo queda integrar dos sensores para que grafiquen datos, respecto al Cansat, ya se encuentra montado en su estructura definitiva y se han realizado las pruebas de transmisión y lectura con buenos resultados.

Misión Primaria: 100%

Misión Secundaria: 80%

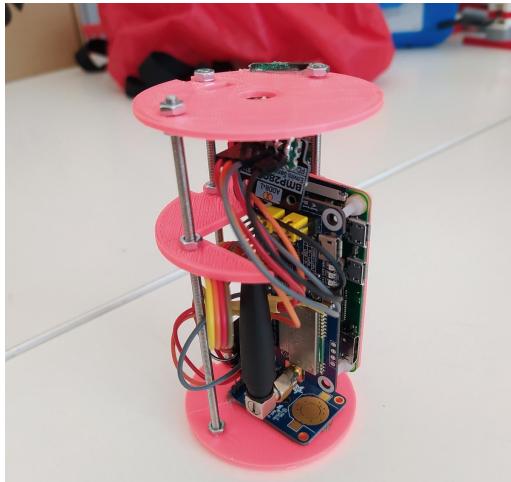


Imagen 24: Cansat Montado más estructura interior

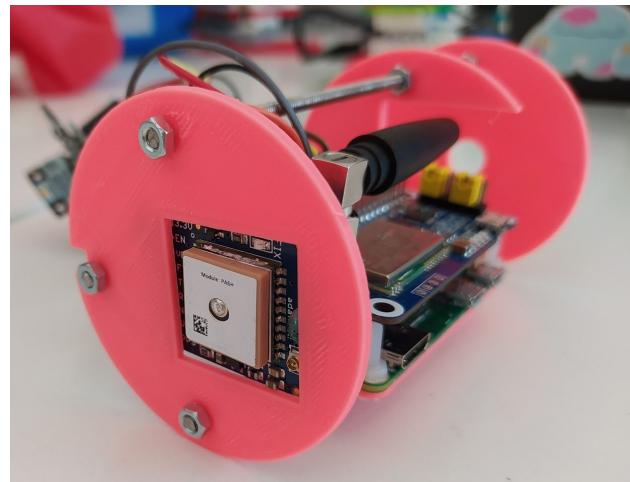


Imagen 25: Detalle de componentes GPS

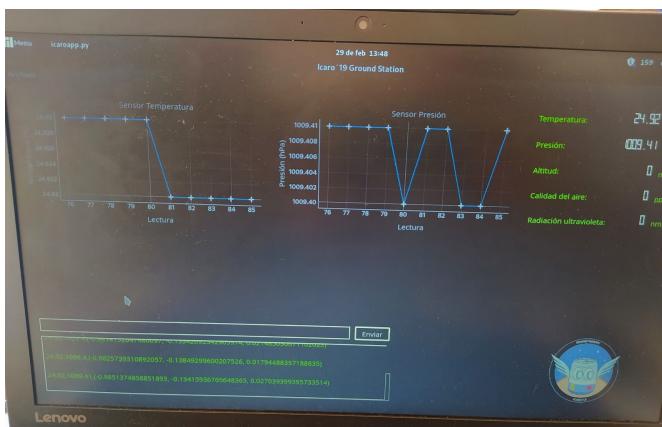


Imagen 26: Estacion de Tierra recibiendo en tiempo real



## ANEXO 1

### Código Cansat

#### Introducción

El desarrollo de la aplicación del Cansat es relativamente sencillo, aún quedan un par de sensores a integrar pero el proceso no parece difícil. Toda la información referente a los sensores y el código que hay que utilizar es amplia y bien documentada ya que todos nuestros sensores son de Adafruit y es fantástico como tienen implementadas las librerías de estos, además disponen de muchos ejemplos que podemos utilizar directamente.

El proyecto está desarrollando sobre una Raspberry Pi Zero o 3 B+. Los sensores utilizados son el BMP280, LSM303D, SGP30 y VEML6075, todos ellos de Adafruit.

#### Código

Importamos las librerías necesarias:

```
import RPi.GPIO as GPIO
import serial
import time
import sys
```

Inicialización de los distintos sensores y preparación para transmitir por LORA a través del puerto serie:

```
# SENSOR SMP280 - temperatura y presión
from bmp280 import BMP280

#SENSOR LSM303D - acelerómetro
from lsm303d import LSM303D
```



```
# Preparar modo de transmisión de datos por lora mediante P2P

M0 = 22
M1 = 27
#MODE = [ "BROADCAST_AND_MONITOR", "P2P" ]

CFG_REG = [b'\xC2\x00\x09\xFF\xFF\x00\x62\x00\x17\x03\x00\x00',
            b'\xC2\x00\x09\x00\x00\x00\x62\x00\x17\x03\x00\x00']
RET_REG = [b'\xC1\x00\x09\xFF\xFF\x00\x62\x00\x17\x03\x00\x00',
            b'\xC1\x00\x09\x00\x00\x00\x62\x00\x17\x03\x00\x00']
r_buff = ""
delay_temp = 1

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(M0,GPIO.OUT)
GPIO.setup(M1,GPIO.OUT)

GPIO.output(M0,GPIO.LOW)
GPIO.output(M1,GPIO.HIGH)
time.sleep(0.01)

ser = serial.Serial("/dev/ttyS0",9600)
ser.flushInput()

# Inicializar el BMP280
try:
    from smbus2 import SMBus
except:
    from smbus import SMBus

bus = SMBus(1)
bmp280 = BMP280(i2c_dev=bus)

# Para calcular la altura debemos realizar primero una serie de
# lecturas y almacenarlas en una lista para calcular la línea base
# mediante una media de estas
```



```
baseline_values = []
baseline_size = 100

print("inicializando linea base...")
for i in range(baseline_size):
    pressure = bmp280.get_pressure()
    baseline_values.append(pressure)
    time.sleep(1)
    print(i, "% calculando linea base")
baseline = sum(baseline_values[:-25]) / len(baseline_values[:-25])

#Inicializar el LSM303D
lsm = LSM303D(0x1d)
```

La siguiente sección de código es el bucle principal del código. Hemos aprovechado el código que ofrece el fabricante del sistema de transmisión por Lora y se le ha añadido la lectura de datos de los distintos sensores para componer el mensaje a transmitir. Las lecturas son actualizadas y se envían cada segundo. Hemos planteado la posibilidad de recepción de mensajes procedentes de otros cansat, así que incluiremos una cabecera a nuestros menajes que los hagan únicos y así poder descartar otros mensajes desde la estación de tierra.

```
try :

    if ser.isOpen() :
        print("It's setting P2P mode")
        ser.write(CFG_REG[1])
        contador = 0

    while True :

        if ser.inWaiting() > 0 :
            time.sleep(0.1)
            r_buff = ser.read(ser.inWaiting())
            if r_buff == RET_REG[1] :
                print("P2P modo activado")
                GPIO.output(M1,GPIO.LOW)
                time.sleep(0.01)
```



```

        r_buff = ""
        if r_buff != "" :
            print("receive a P2P message:")
            print(r_buff)
            r_buff = ""

delay_temp += 1

if delay_temp > 10000 :
    # LEER DATOS
    temperature = bmp280.get_temperature()
    pressure = bmp280.get_pressure()
    altitude = bmp280.get_altitude(qnh=baseline)
    xyz = lsm.accelerometer()
    accel = "{:06.2f}:{:06.2f}:{:06.2f}".format(*xyz)

        #Componemos el mensaje a transmitir con la lecturas. Debemos
#incluir cabecera en el mensaje identificarlo que es nuestro en la estación
#de tierra
    mensaje = str(contador)+ "," + str(round(temperature,2)) + ","
+ str(round(pressure,2)) + "," + str(round(altitude,2)) + "," + accel +
"\r\n"

# Enviamos el por LORA a través del puerto serie
    ser.write(mensaje.encode())
    delay_temp = 0
    contador += 1

except :
    if ser.isOpen() :
        ser.close()
    GPIO.cleanup()

```

Todo el código se encuentra disponible en nuestro repositorio de GitHub, el cual se irá actualizando cuando se incorporen los componentes que faltan (sensor de calidad de aire, sensor de rayos ultravioleta y GPS).



## Código Ground Station

### Introducción

El desarrollo de esta aplicación es un poco más complejo que la del satélite debido sobre todo, a la forma en la que hay que estructurar las aplicaciones gráficas. Aunque cada librería gráfica tiene sus peculiaridades, todas coinciden en la estructura de control de la aplicación. La parte más compleja ha sido averiguar la forma de construir un subprocesso que sea el encargado que leer el puerto serie. Este problema es debido a que las aplicaciones gráficas funcionan conectando eventos que sucedan dentro de la aplicación con funciones que realicen una acción, por lo que si una función se bloquea o requiere de cierta cantidad de tiempo para realizar la acción, la aplicación quedaría bloqueada hasta que se completara esa acción.

Tras consultar la documentación y enlaces sobre páginas dedicadas a PyQt5, optamos por la creación de un subprocesso que se encargue de leer el puerto serie y se comunique por medio de señales con la aplicación para poder actualizar los datos, de esta forma minimizamos el riesgo de bloquear la aplicación principal.

Dentro del GitHub del proyecto (incluimos el enlace en la biografía), en la carpeta correspondiente a Ground Station se encuentran varios archivos:

**README.md** → Descripción del proyecto, instrucciones y librerías necesarias.

**ground\_station.ui** → Ventana de la aplicación gráfica generada con Qt Designer.

**ground\_station.py** → Conversión del archivo .ui para ser utilizado con python mediante pyuic5.




---

**icaroapp.py** → Aplicación principal de Ground Station. Utiliza *ground\_station.py*

**serialThreadFile.py** → Se encarga de la lectura del puerto serie para la recepción del mensaje por LORA y se comunica con *icaroapp.py* mediante señales por las que pasa la información.

## Código

A continuación mostramos el código que pertenece a *ground\_station.py*, *icaroapp.py* y *serialThreadFile.py*. El archivo *ground\_station.ui* es el archivo generado desde la aplicación Qt Designer, que utilizamos para el diseño de la ventana principal de forma visual. La conversión del archivo .ui a .py la realizamos con el comando **pyuic5** disponible en PyQt5 desde el terminal.

### **ground\_station.py**

No vamos a mostrar todo su contenido debido a que no es un código que hayamos desarrollado nosotros si no que ha sido generado por la conversión, a parte es demasiado extenso y su información es susceptible de cambios, ya que cada vez que cambiemos el diseño éste archivo cambiará. Es una buena práctica hacerlo de esta forma ya que se separa la parte de diseño de la de programación.

Esta parte del código es la más relevante, ya que es la clase que utilizaremos en nuestra aplicación como herencia para generar la ventana principal. Esta es la parte más abstracta ya que corresponde a la programación orientada a objetos, y las aplicaciones gráficas hacen bastante uso de ella.

```
# -*- coding: utf-8 -*-

# Form implementation generated from reading ui file 'ground_station.ui'
#
# Created by: PyQt5 UI code generator 5.14.1
#
# WARNING! All changes made in this file will be lost!
```



```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1360, 728)
```

## icaroapp.py

Este es el código de nuestra aplicación gráfica. Utilizamos *ground\_station.py* para generar la ventana. Entre otras librerías que son necesarias, la que más puede destacar es **pyqtgraph** que es la que se encarga de graficar los datos.

Importar las librerías necesarias:

```
from PyQt5.QtGui import *
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from ground_station import *
import pyqtgraph as pg

from serialThreadFile import serialThreadClass

import sys
```

El siguiente código pertenece a la clase principal que utilizamos para construir la ventana de la aplicación utilizando la herencia para construirla. El archivo *serialThreadFile.py* se utiliza para crear un proceso paralelo que se comunique con la aplicación principal mediante señales. Para actualizar las gráficas hemos creado



un temporizador “Timer” que actúa cada segundo conectado a la función que actualiza la lista de valores del sensor a graficar. Por motivos de optimización no redibujamos la curva con todos los valores, si no con los 10 últimos, esto podemos ajustarlo cambiando el tamaño de la lista para almacenar los datos.

```
class MyWindow(QMainWindow, Ui_MainWindow):
    def __init__(self):
        super().__init__()

        self.setupUi(self)

        self.lectura = []

        self.mySerial = serialThreadClass()
        self.mySerial.mensaje.connect(self.leer_serial)

        self.mySerial.start()

        self.x = list(range(10))
        self.y = [0 for _ in range(10)]

        self.temperature_y = [0 for _ in range(10)]
        self.presure_y = [0 for _ in range(10)]
        pen = pg.mkPen(color=(0, 100, 255))

        self.graphicsConfig(self.graphicsTemperature, "Sensor
Temperatura", "Temperatura (°C)", "Lectura", self.temperature_y)
        self.data_line_temp = self.graphicsTemperature.plot(self.x,
self.temperature_y, pen=pen, symbol="+")

        self.graphicsConfig(self.graphicsPresure, "Sensor Presión",
"Presión (hPa)", "Lectura", self.presure_y)
        self.data_line_pres = self.graphicsPresure.plot(self.x,
self.presure_y, pen=pen, symbol="+")

    self.timer = QtCore.QTimer()
    self.timer.setInterval(1000)
    self.timer.timeout.connect(self.update_plot_data)
```



```

        self.timer.start()

def graphicsConfig(self, graphics,sensor, lbl_y, lbl_x, lista):
    graphics.setYRange(0, 40, padding=0)
    graphics.showGrid(x=True, y=True)
    graphics.setTitle(sensor)
    graphics.setLabel('left', lbl_y)
    graphics.setLabel('bottom', lbl_x)

def update_plot_data(self):
    self.x = self.x[1:] # Eliminamos el primer elemento de la lista
    self.x.append(self.x[-1] + 1) # Añadimos nuevo elemento sumando 1
al anterior
    self.data_line_temp.setData(self.x, self.temperature_y) #
Actualizamos la curva de la grafica
    self.data_line_pres.setData(self.x, self.pressure_y)

def leer_serial(self, msg):
    self.textEdit_2.append(msg)
    self.temperature_y = self.temperature_y[1:] # Eliminamos el
primer valor de la lista
    self.pressure_y = self.pressure_y[1:]

    self.lectura = msg.split(",")

    self.lcdTemperature.display(float(self.lectura[1]))
    self.lcdPressure.display(float(self.lectura[2]))
    self.lcdAltitude.display(float(self.lectura[3]))

try: # si no hay lectura desde el cansat agregamos a la lista el
valor 0
    self.temperature_y.append(float(self.lectura[1]))
    self.pressure_y.append(float(self.lectura[2]))
except ValueError:
    print ("Valor erroneo")
    self.temperature_y.append(0)
    self.pressure_y.append(0)

```



```

def plot(self, h, t):
    self.graphicsTemperature.plot(h,t)
    self.graphicsPresure.plot()

# Código que se encarga de crear la ventana y ejecutar la aplicación
app = QApplication(sys.argv)
window = MyWindow()
window.show()
app.exec_()

```

### **serialThreadFile.py**

El siguiente programa lo utilizamos como proceso independiente al de la aplicación para que esté leyendo del puerto serie cada segundo y decodificando los datos para transmitirlos a la aplicación mediante una señal. Hemos decidido modificar esta parte para poder diferenciar los mensajes de nuestro cansat y controlar fallos de lectura que puedan ocurrir y por tanto paralizar la aplicación. Según las especificaciones de PyQt5 para crear un proceso debemos hacerlo mediante una clase que herede de QThred, la cual dispone de funciones para llamar e inicializar el proceso.

```

import serial
from PyQt5.QtCore import pyqtSignal, QThread
import time

# Nuestra clase hereda de QThread, necesaria para crear un proceso
class serialThreadClass(QThread):
    mensaje = pyqtSignal(str) # Creamos la señal para comunicarnos con la

```



```
# aplicación

def __init__(self, parent = None):
    super().__init__(parent) # Constructor del proceso
    # Inicializamos el puerto serie
    self.ser = serial.Serial('/dev/ttyUSB0', 9600)
    time.sleep(2)

def run(self):
    while True:

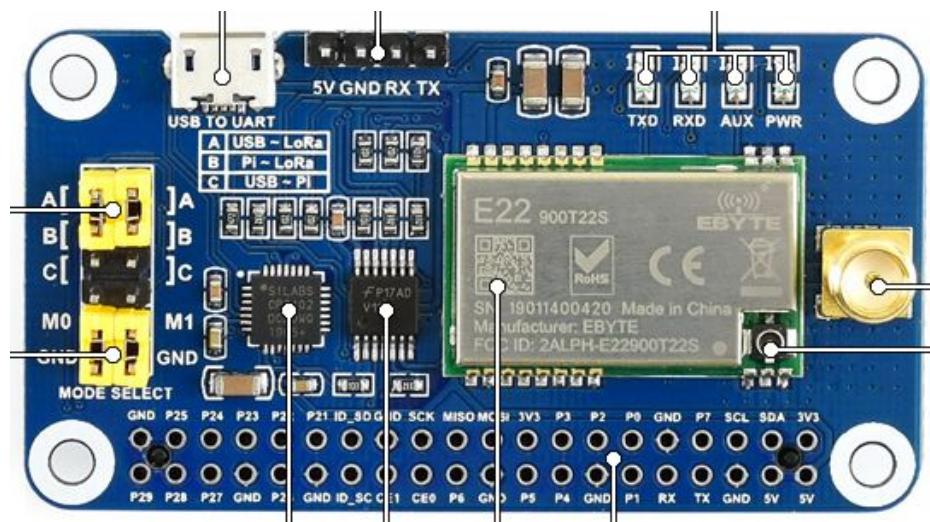
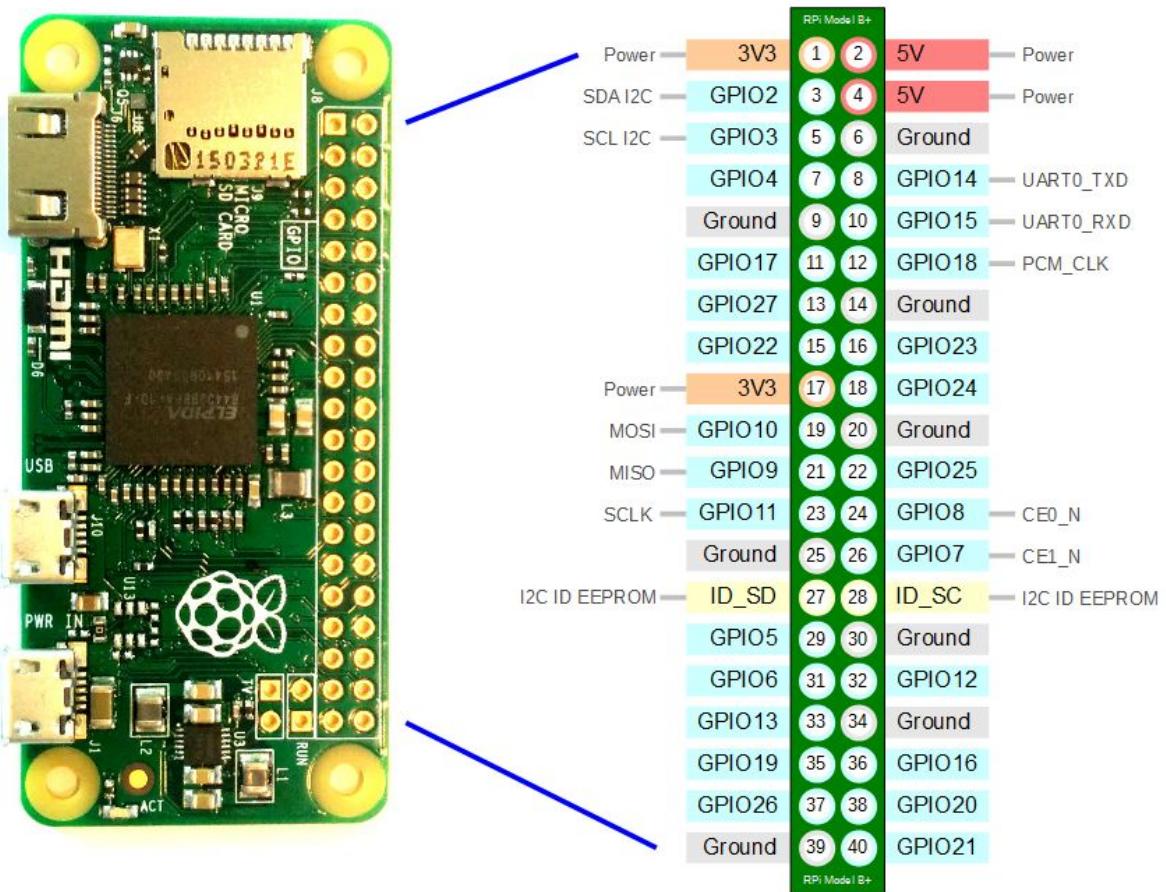
        lectura = self.ser.readline()
        self.mensaje.emit(str(lectura.decode())) # Pipe con la
aplicacion
        print (lectura)
        time.sleep(1)

def sendSerial(self):
    self.ser.write(b'A')
```



## ANEXO 2

### PINOUT





| Nombre   | IMG   | PIN OUT  |
|----------|---|--|
| LSM303D  | A small black PCB with a blue component labeled "LSM303D" and several gold pins.  | GND (6) /<br>3-6V(1) / SDA<br>(3) / SCL (5) /<br>INT (7) |
| BMP280   | A small black PCB with a blue component labeled "BMP280" and several gold pins.   | GND (6) /<br>3-6V(1) / SDA<br>(3) / SCL (5)              |
| SGP30    | A small black PCB with a blue component labeled "SGP30" and several gold pins.    | GND (6) /<br>3-6V(1) / SDA<br>(3) / SCL (5)              |
| VEML6075 | A small black PCB with a blue component labeled "VEML6075" and several gold pins. | GND (6) /<br>3-6V(1) / SDA<br>(3) / SCL (5)              |
| GPS      | A small blue PCB with a white GPS chip and several gold pins.                     | VIN (2) / GND<br>(14) / RX(8) /<br>TX (10)               |



---

## BIBLIOGRAFÍA

**GitHub del proyecto:** <https://github.com/icaro19mur/Icaro19.git>

**Python:** <https://www.python.org/>

**PyQt5:** <https://www.riverbankcomputing.com/software/pyqt/download5>

**PySide2:** <https://maven.apache.org/>

**Tutoriales PyQt5:** <https://www.learnpyqt.com/>

**GitHub Sensor temperatura y presión:**

[https://github.com/adafruit/Adafruit\\_CircuitPython\\_BMP280](https://github.com/adafruit/Adafruit_CircuitPython_BMP280)

**GitHub Sensor acelerómetro y magnetómetro LSM303d:**

[https://github.com/adafruit/Adafruit\\_CircuitPython\\_LSM303\\_Accel](https://github.com/adafruit/Adafruit_CircuitPython_LSM303_Accel)

**GitHub Sensor calidad del aire CO2 SGP30:**

[https://github.com/adafruit/Adafruit\\_CircuitPython\\_SGP30](https://github.com/adafruit/Adafruit_CircuitPython_SGP30)

**GitHub Sensor radiación ultravioleta VEML6075:**

[https://github.com/adafruit/Adafruit\\_CircuitPython\\_VEML6075](https://github.com/adafruit/Adafruit_CircuitPython_VEML6075)

**LORA HAT:** [https://www.waveshare.com/wiki/SX1262\\_868M\\_LoRa\\_HAT](https://www.waveshare.com/wiki/SX1262_868M_LoRa_HAT)

