UNIVERSIDAD DE GRANADA

Facultad de Ciencias

GRADO EN INGENIERÍA
ELECTRÓNICA INDUSTRIAL

TRABAJO FIN DE GRADO

PLATAFORMA DE CONTROL ELECTRÓNICO DE MISIÓN PARA CUBESAT SOBRE COSMOS

Presentado por:
D. Antonio Ortiz González

Tutor:
Prof. Andrés María Roldán Aranda

Curso académico 2024/2025

UNIVERSIDAD
DE GRANADA

GRADO EN INGENIERIA ELECTRÓNICA INDUSTRIAL

PLATAFORMA DE CONTROL ELECTRÓNICO DE MISIÓN PARA CUBESAT SOBRE COSMOS

Autor: Antonio Ortiz González

Directores: Andrés María Roldán Aranda

Departamento: Electrónica y tecnología de computadores

Palabras clave: CubeSat, Estación Terrena, OBC, ADCS, COSMOS, Diseño Aeroespacial, SolidWorks, Ingeniería inversa, Ingeniería de sistemas.

Resumen: Este Trabajo Fin de Grado tiene como objetivo aplicar los conocimientos adquiridos durante el grado y ampliar conocimientos en ingeniería espacial, siguiendo una metodología profesional propia del sector. Se desarrollará en el laboratorio aeroespacial GranaSAT, abordando mejoras en hardware y software del CubeSat y añadiendo funcionalidades de comunicación con estación terrena y control de actitud.

Índice

Índice de Figuras

Índice de Tablas

Índice de Videos

Glosario

actitud Orientación espacial de un satélite, definida por cómo se posiciona respecto a un sistema de referencia, como la Tierra, el Sol o las estrellas.

carga útil Conjunto de equipos o instrumentos de un satélite destinados a cumplir su misión principal, como cámaras, sensores o transpondedores.

colimador Dispositivo que restringe y dirige un haz de partículas o radiación para mejorar su precisión y enfoque.

cámara termográfica Dispositivo que detecta y visualiza la radiación infrarroja emitida por los objetos para medir su temperatura.

diagrama de Gant Herramienta de gestión de proyectos que representa visualmente el cronograma de actividades y tareas a lo largo del tiempo.

estación terrena Instalación en la Tierra equipada con antenas y sistemas de comunicación para enviar y recibir datos desde satélites u otras naves espaciales.

GranaSAT GranaSAT es un proyecto académico de la Universidad de Granada que originalmente se concibió para diseñar y desarrollar un picosatélite (CubeSat). Coordinado por el Profesor Andrés María Roldán Aranda, GranaSAT se ha convertido en un proyecto multidisciplinar en el que hoy en día participan estudiantes de diferentes titulaciones, permitiéndoles adquirir y ampliar los conocimientos en áreas como la ingeniería aeroespacial o la física de partículas.

Magnetopares Actuadores que generan torque en un satélite al interactuar con el campo magnético terrestre para controlar su orientación.

MobaXterm Herramienta que proporciona un terminal avanzado para tareas de administración remota, integrando clientes SSH, SFTP, y múltiples utilidades para trabajar en entornos Unix desde Windows.

Power Analyzer Instrumento que mide y analiza parámetros eléctricos como voltaje, corriente, potencia y eficiencia en sistemas eléctricos.

SolidWorks Software de diseño asistido por computadora (CAD) que permite modelar y simular piezas, ensamblajes y dibujos en 3D para ingeniería y fabricación.

volante de inercia Dispositivo mecánico utilizado en satélites para controlar y estabilizar su actitud mediante la conservación del momento angular.

Acrónimos

ADC Analog-Digital Converter.

ADCS Attitude Determination & Control System.

control PID control Proporcional Integral Derivativo.

ECSS European Cooperation for Space Standardization.

FPGA Field-Programmable Gate Array.

GPIO General Purpose Input Output.

I2C Inter Integrated Circuit.

I2DOS Inertial 2D Orbital Simulator.

IC Integrated Circuit.

LAN Local Area Network.

LDR Light Dependent Resistor.

LED Light Emitting Diode.

NASA National Aeronautics and Space Administration.

OBC On-Board Computer.

PCB Printed Circuit Board.

PWM Pulse Width Modulation.

RAM Random Access Memory.

RTC Real Time Clock.

SBC Single Board Computer.

SPI Serial Peripheral Interface.

SSH Secure Shell.

UDP User Datagram Protocol.

USB Universal Serial Bus.

Capítulo 1

Introducción

1.1 Motivación

Este Trabajo Fin de Grado se plantea con el objetivo de reunir y aplicar los conocimientos adquiridos a lo largo de los cuatro años de grado a la vez que ampliar los mismos en diferentes áreas en las que no haya tenido la oportunidad de adentrarme. Todo ello aplicado a un sector muy demandante técnicamente como es el de la ingeniería espacial, con un enfoque y metodología propias del trabajo en una empresa. Otra gran oportunidad que brinda este TFG es que su realización se lleva a cabo en su totalidad en el laboratorio de electrónica aeroespacial de GranaSAT, lo que permitirá trabajar con equipo y herramientas a las que no había tenido antes acceso, todo ello mientras me sumerjo en el ambiente de trabajo interdisciplinar propio de un laboratorio.

Ref.	Motivación
Mot.1	Familiarizarse con la metodología de desarrollo de proyectos.
Mot.2	Aprender sobre los estándares y requerimientos técnicos propios del sector aeroespacial.
Mot.3	Aplicar el conocimiento adquirido durante el grado en Ingeniería Electrónica Industrial.
Mot.4	Ampliar dicho conocimiento abordando problemas fuera del ámbito de la electrónica industrial, tales como las telecomunicaciones o la ingeniería mecánica.
Mot.5	Abordar un proyecto real de ingeniería con la metodología propia del sector empresarial.
Mot.6	Generar documentación sobre todo el procedimiento con el fin de facilitar el trabajo a futuros estudiantes que deseen seguir con el proyecto.
Mot.7	Superar la asignatura "Trabajo fin de Grado".

Tabla 1.1 – *Motivaciones académicas y profesionales*

Figura 1.1 – *Logo de GranaSAT*

1.2 Descripción de Capítulos

De acuerdo con la metodología general de desarrollo de proyectos y concretando al sector aeroespacial se plantea la siguiente división en capítulos:

- **Capítulo 2: Estudio Previo**
  Este capítulo se centrará en presentar el estándar CubeSat y describir el estado en el que se dio por finalizado el TFM en el que se basa este proyecto con el fin de identificar las necesidades y líneas de mejora presentes en el mismo.
- **Capítulo 3: Requerimientos**
  En este capítulo se redactarán los requerimientos técnicos y funcionales de cada uno de los tres sistemas principales del proyecto, que guiarán su posterior desarrollo a lo largo de las posteriores fases de análisis y diseño.
- **Capítulo 4: Planificación y Presupuesto**
  Este capítulo describe las herramientas de gestión utilizadas en la planificación del periodo de vida del proyecto y la redacción de un presupuesto que evaluaría el coste de la realización de este proyecto en una empresa privada.
- **Capítulo 5: Análisis.**
  En este capítulo se estudiarán las diferentes soluciones que permitan satisfacer los requerimientos definidos. Una vez elegida una solución se planteará un esquema funcional para cada uno de los sistemas, que se materializará posteriormente en la fase de Diseño de Sistema.
- **Capítulo 6: Diseño de Sistema.**
  Después de las distintas etapas de análisis e ingeniería realizadas en los capítulos anteriores, este capítulo abordará el diseño del sistema, describiendo tanto la materialización de las soluciones planteadas como las nuevas funcionalidades programadas.
- **Capítulo 7: Validación.**
  Este capítulo se utiliza para validar las soluciones propuestas en esta Trabajo Fin de Grado y verificar el cumplimiento de los requerimientos del proyecto.
- **Capítulo 8: Conclusión y Futuras Líneas de Trabajo.**
  Finalmente se revisará el trabajo realizado y se identificaran líneas de mejora que puedan servir como punto de partida a futuros proyectos.

Capítulo 2

Estudio Previo

2.1 ¿Qué es un CubeSat?

CubeSat es un estándar de diseño de nanosatélites especificado, por las universidades de Cal Poly y Stanford [7], como resultado del elevado coste de operación en el ámbito espacial, donde se necesitan grandes cantidades de dinero para llevar a cabo misiones con un riesgo considerable. Los CubeSats permitieron sacrificar algunas funcionalidades a cambio de una reducción significativa de los costes, lo que permite un mayor número de misiones.

A pesar de que durante muchos años su principal función era académica, son muchas las empresas que ahora los utilizan con fines de investigación, como mapeo de la tierra, pruebas de tecnología en el espacio o funciones de telecomunicaciones. Incluso pueden destacarse importantes misiones realizadas por la NASA que utilizaron el estándar CubeSat, ejemplo de ello son los satélites **MarCO-A** y **MarCO-B** [5], apodados Wall-E y Eva, que se lanzaron el 5 de mayo de 2018 con el objetivo de sobrevolar Marte y proporcionar comunicaciones en tiempo real mientras el aterrizador fijo **InSight** [4] realizaba la fase de entrada, descenso y aterrizaje.

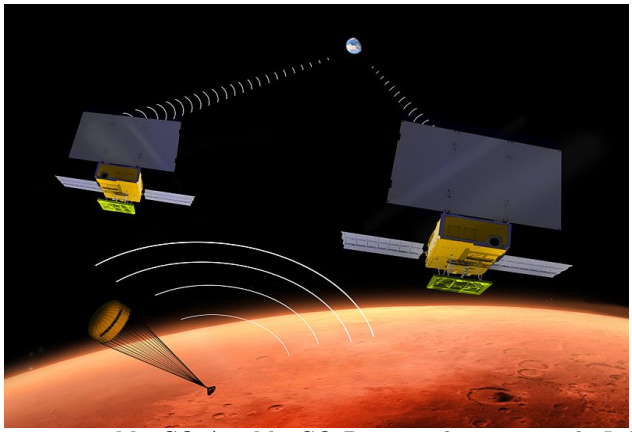


Figura 2.1 – *MarCO-A y MarCO-B junto al aterrizaje de InSight*

Los CubeSats están diseñados para ser ligeros, económicos y compatibles con lanzadores convencionales, lo que permite que sean transportados al espacio como carga secundaria en lanzamientos de cohetes más grandes. En el estándar CubeSat se define una unidad (1U) [7] como un cubo de 10x10x10 cm con un peso máximo de 1.33 kg. Retomando a Wall-E y Eva, ambos eran CubeSats de formato 6u.

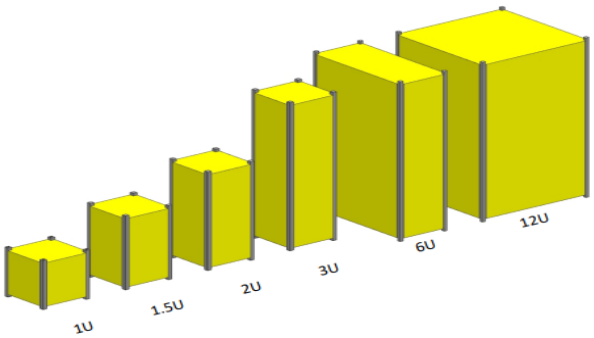


Figura 2.2 – *Familia CubeSat*

2.2 European Cooperation for Space Standardization

Con el fin de plantear el desarrollo de este proyecto con la metodología propia del sector aeroespacial se seguirá el estándar **ECSS-E-ST-10C** [2] de la ECSS.

La ECSS es una iniciativa establecida para desarrollar un conjunto coherente y único de estándares fáciles de usar para todas las actividades espaciales en Europa. Comenzó oficialmente en el otoño de 1993, cuando la comunidad espacial europea se dio cuenta de que las diferencias en los estándares espaciales resultaban en mayores costes, menor eficacia y una industria menos competitiva.

El estándar **ECSS-E-ST-10C** [2] describe los requerimientos generales de un proyecto de ingeniería de sistemas en el ámbito espacial.

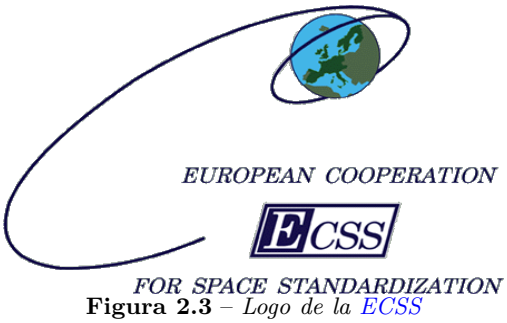


Figura 2.3 – *Logo de la ECSS*

2.3 Prototipo GranaSat-I

Este TFG continúa con el proyecto iniciado en 2018 por el alumno José Carlos Martínez Durillo como parte de su Trabajo fin de Máster, titulado "Design of a multidisciplinary 1U CubeSat Simulation Platform". José Carlos fue el encargado de diseñar y desarrollar tanto el Satélite como el entorno de pruebas que hoy nos permiten poder llevar a cabo este trabajo.

Sin embargo, al centrarse su proyecto en el desarrollo del propio satélite a nivel físico, no se pudo profundizar en la programación del algoritmo principal que rija el comportamiento del mismo, ni dotarlo de ninguna funcionalidad específica o propósito.

Es por eso que hoy, seis años después, tomamos el relevo de José Carlos con el fin de ahondar en su trabajo con dos objetivos en mente. Primero estudiar, revisar y mejorar diferentes aspectos tanto del hardware como del software del CubeSat y segundo, crear nuevas funcionalidades de comunicación y control de actitud.

Siguiendo la estructura que ya propuso José Carlos, dividiremos este proyecto en tres sistemas diferentes, la plataforma mecánica de pruebas, la estación terrena y el propio CubeSat.

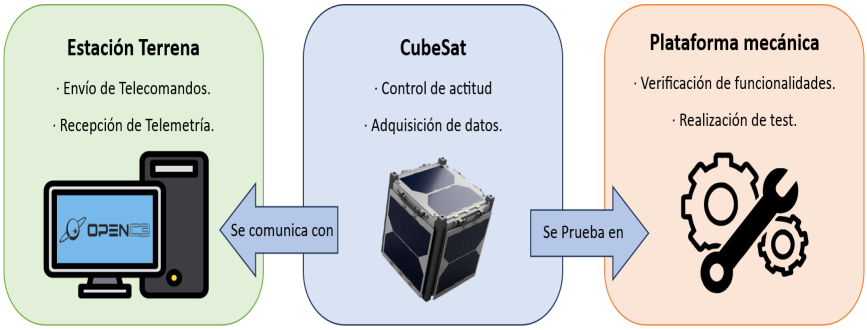


Figura 2.4 – *Sistemas principales del proyecto*

2.3.1 Plataforma Mecánica

La **plataforma mecánica** se ideó con el fin de proporcionar un espacio de simulación y testeo reconfigurable para CubeSats de 1U. Consta principalmente de dos partes, el I2DOS y los **simuladores solares**. La misión del I2DOS será testear el comportamiento del satélite en un ambiente lo más similar posible a estar en una órbita real. Mientras tanto, los simuladores solares, acoplados en la misma plataforma mecánica que el I2DOS, están formados por potentes LEDs que simularán la llegada de luz solar al satélite.

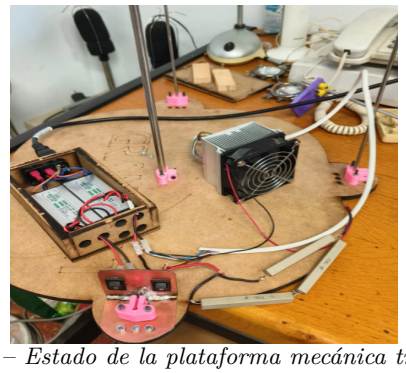


Figura 2.5 – *Estado de la plataforma mecánica tras seis años*

2.3.2 Estación Terrena

Una estación terrena es una pieza clave en el control de misión ya que permiten el envío de **telecomandos**, el seguimiento del satélite y el procesamiento de datos y **telemetría**.

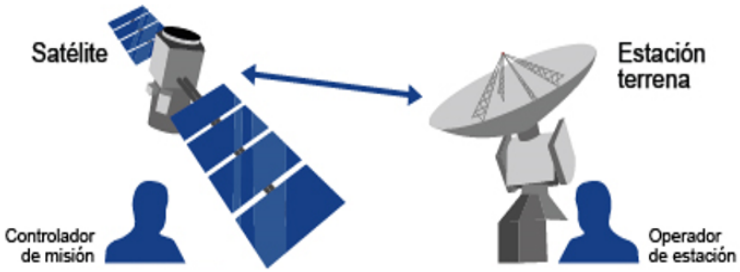


Figura 2.6 – *Propósito de una estación terrena*

2.3.2.1 OpenC3 COSMOS

Concretando a este proyecto, se utilizará el software COSMOS como base para el diseño de la estación terrena. **OpenC3** es una empresa fundada en junio de 2022 por los creadores de Ball Aerospace COSMOS con el objetivo de ofrecer al mundo una herramienta que permita a cualquier persona conectar fácilmente diferentes piezas de hardware. Principalmente trabajan para la industria aeroespacial en la integración y pruebas de satélites en sus operaciones, pero en cualquier lugar donde haya hardware embebido COSMOS puede ofrecer una excelente plataforma para integrar todas las piezas y "reunirlo todo".

Figura 2.7 – *Logo de OpenC3 COSMOS*

2.3.3 CubeSat

El CubeSat conforma el corazón de este proyecto y el sistema más complejo. Su estructura puede subdividirse en tres bloques principales: el OBC, el ADCS y la carga útil.

El OBC es el cerebro del sistema y es el encargado de coordinar el resto de subsistemas. Está conformado por una **raspberry CM3**, uno de las opciones más conocidas entre los SBCs. Cuenta con un chip Broadcom BCM2837 con hasta 1.2 GHz, 1 GB de RAM, 4 GB de memoria flash y 45 GPIOs.

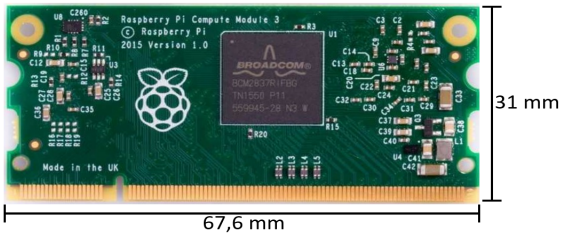


Figura 2.8 – *Raspberry Compute Module 3*

Compartiendo PCB con el OBC se integran varios ICs dedicados a las interfaces de comunicación tanto cableadas como inalámbricas, entre los que destaca un chip **ESP8266** esclavo del OBC exclusivo para la **comunicación WiFi**. También encontramos diferentes sensores o actuadores que conforman la carga útil y se listan a continuación:

(1) Acelerómetro, Giroscopio y Magnetómetro (MPU9250).

(2) Barómetro y sensor de Temperatura (Bosch BMP280).

(3) 2 ADCs de 12 bits (ADS1015IDGST).

(4) Puente H (ST-L298N).

(5) Tacómetro (CNY70).

Por último, el ADCS es el subsistema encargado del control de la actitud del CubeSat, una tarea clave en un satélite ya que de ella dependen funciones cruciales como pueden ser orientarse hacia el sol para cargar baterías con energía solar o posicionarse correctamente para medir con los sensores aquellas magnitudes que sean objeto de misión. Para ello el ADCS de nuestro CubeSat cuenta con una FPGA ICE40UP5K de la empresa **Lattice** que actuará como esclavo del OBC y se encargará de controlar los sistemas de posicionamiento.

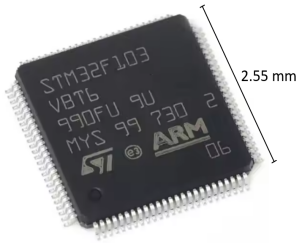


Figura 2.9 – *FPGA ICE40UP5K de Lattice*

Capítulo 3

Requerimientos

3.0.1 Plataforma Mecánica

Ref.	Requerimiento
PM.Req.1	Reensamblaje del I2DOS y sustitución de posibles piezas deterioradas.
PM.Req.2	Caracterización de los LEDs implementados en los simuladores solares.
PM.Req.3	Búsqueda de un método de alimentación que permita la operación de forma segura.
PM.Req.4	Diseño de un sistema de ventilación para la refrigeración de los simuladores solares.

Tabla 3.1 – *Plataforma mecánica - Requerimientos*

3.0.2 Estación Terrena

Ref.	Requerimiento
ET.Req.1	Diseño del panel de control de misión.
ET.Req.2	Definición de paquetes de telemetría que permitan la transmisión de la información de los sensores del CubeSat.
ET.Req.3	Definición de un listado de telecomandos que permitan alterar el comportamiento y actitud del CubeSat.
ET.Req.4	Diseño de una interfaz de comunicación estable y fiable tanto cableada como inalámbrica que permita integrar uno o varios CubeSats.

Tabla 3.2 – *Estación terrena - Requerimientos*

3.0.3 CubeSat

Ref.	Requerimiento
CS.Req.1	Reposición o sustitución de sensores u otros elementos no implementados o defectuosos.
Cs.Req.2	Programación de un código que genere y envíe paquetes de telemetría en tiempo real a partir de los datos de los sensores.
CS.Req.3	Programación de un código de recepción e interpretación de telecomandos.
CS.Req.4	Establecimiento de las interfaces de conexión necesarias para la programación del OBC.
Cs.Req.5	Establecimiento de las interfaces de conexión necesarias para establecer una comunicación cableada estable y fiable con la estación terrena.
Cs.Req.6	Restablecimiento de la comunicación inalámbrica con la estación terrena mediante la ESP8266.
CS.Req.7	Programación de un algoritmo que identifique la dirección de la que proviene la luz solar.
CS.Req.8	Programación de un código que permita controlar el volante de inercia para reposicionar el satélite tanto manualmente como de forma autónoma.
CS.Req.9	Diseño de un soporte que resista el enorme par generado por el volante de inercia.
CS.Req.10	El CubeSat debe consumir menos de 5 W en operación.

Tabla 3.3 – *CubeSat - Requerimientos*

Capítulo 4

Planificación y Presupuesto

4.1 Plan de acción

Para el correcto desarrollo de cualquier proyecto es crucial elaborar un **plan de acción** que permita ordenar cronológicamente sus diferentes etapas. En nuestro caso es especialmente importante hacer una buena planificación ya que, al abarcar el tiempo de vida esperado del proyecto desde febrero hasta noviembre, habrá varias semanas en julio, agosto y septiembre, en las que el laboratorio permanecerá cerrado.

Se divide el tiempo de vida total del proyecto en secciones de trabajo de dos semanas y se elabora un diagrama de Gant para organizar las diferentes tareas, de esta manera podemos identificar aquellas tareas que son dependientes de la finalización de otras y aquellas que pueden llevarse a cabo paralelamente.

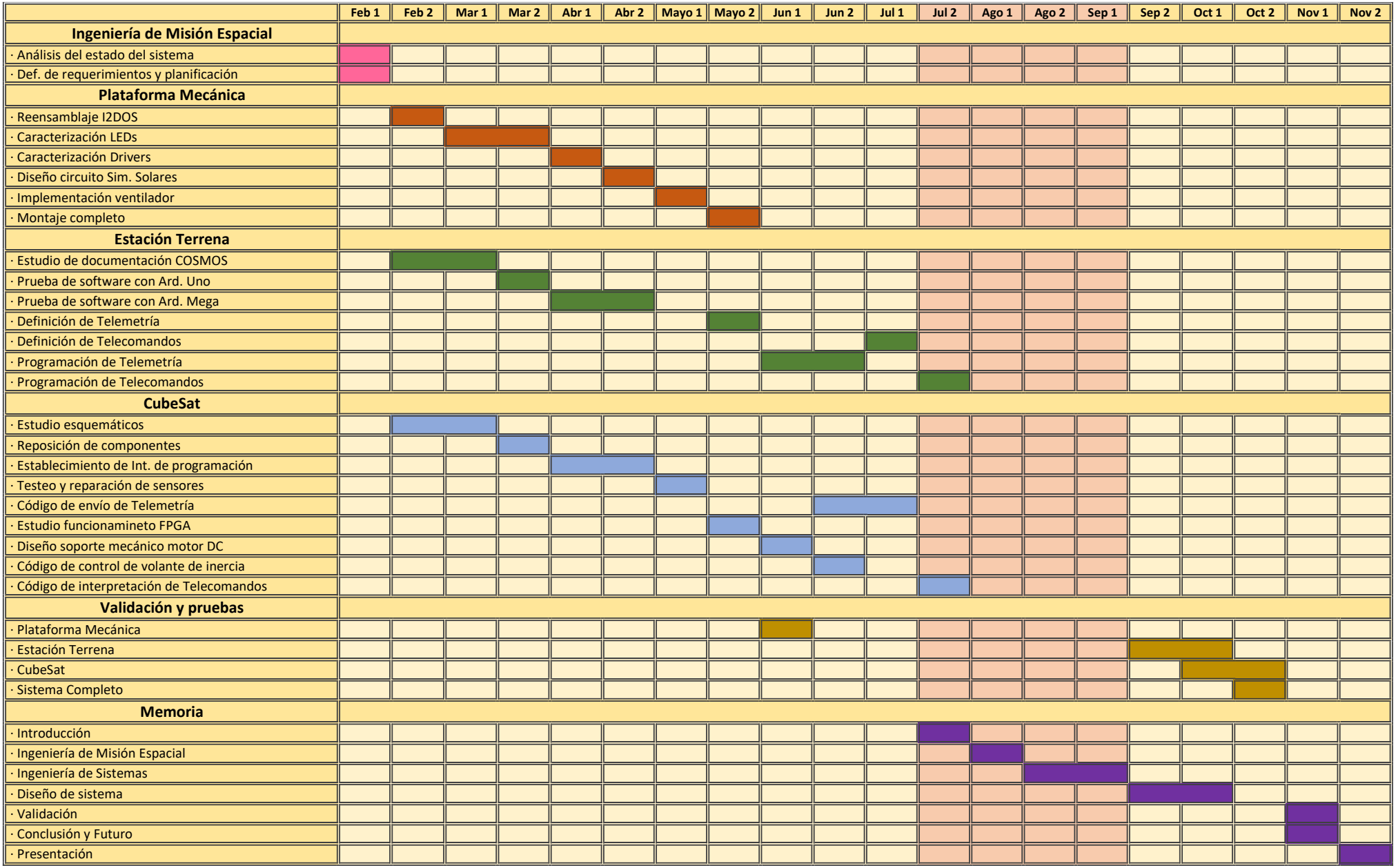


Diagrama de Gant

4.2 Presupuesto

Para concluir este primer capítulo de ingeniería de misión espacial, una vez evaluado el estado general del proyecto y definidos los requerimientos técnicos y funcionales, se procede a la elaboración de un **presupuesto** que evaluaría el coste de la realización de este proyecto en una empresa privada.

componentes	**Coste total [€]**
2 x Resistencias de potencia	2, 08
2 x Ventiladores 12V	0, 80
6 x Resistencias LDR	1, 20
RTC PCF8523	1, 24
Motor DC	2, 40
Rollo cable 0,50 mm^2	11, 78
Filamento impresora 3D	14, 24
Presupuesto	33, 74

Tabla 4.1 – *Presupuesto de componentes*

Recursos Humanos	**Coste total [€]**
Ingeniero Junior (200h)	3000, 00
Ingeniero Senior (20h)	600, 00
Presupuesto	3600, 00

Tabla 4.2 – *Presupuesto de recursos humanos*

Presupuesto Total	**Coste total [€]**
Material	33, 74
Recursos humanos	3600, 00
Gastos indirectos (electricidad y equipo)	200, 00
Margen industrial (10%)	766, 75
Presupuesto	4600.49

Tabla 4.3 – *Presupuesto Total*

Capítulo 5

Análisis

5.1 Plataforma Mecánica

5.1.1 I2DOS

La **plataforma inercial** sobre la que se colocará el CubeSat cuenta con una base soporte para sujetar el satélite, unida mediante cuatro varillas de aluminio a un **anillo inercial** que ayudará a simular el momento de inercia que el CubeSat enfrentaría una vez desplegado en órbita. Toda la estructura estará apoyada mediante un pequeño tornillo en una quinta barra de aluminio unida a la base de la **plataforma mecánica** permitiéndole moverse libremente y sin apenas rozamiento.

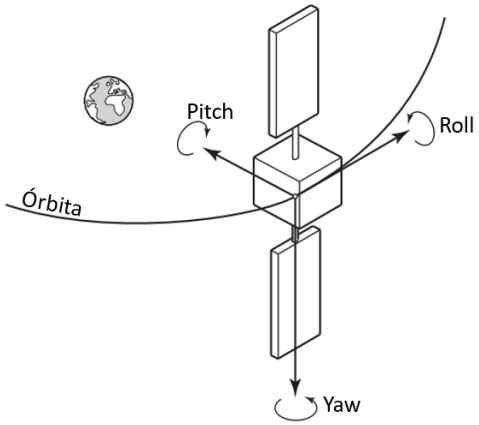


Figura 5.1 – *Movimientos de un satélite en órbita [3]*

Como su propio nombre indica, el I2DOS está diseñado para simular una órbita en 2D. El plano que define la base soporte del CubeSat es perpendicular al eje z y el movimiento objetivo del estudio será el **Yaw**. Este es el movimiento que deberá controlarse mediante la acción del volante de inercia.

5.1.2 Simuladores solares

El objetivo principal de los Simuladores solares es ofrecer una **irradiancia** lo más similar posible a la luz solar, cuyo espectro puede consultarse a continuación:

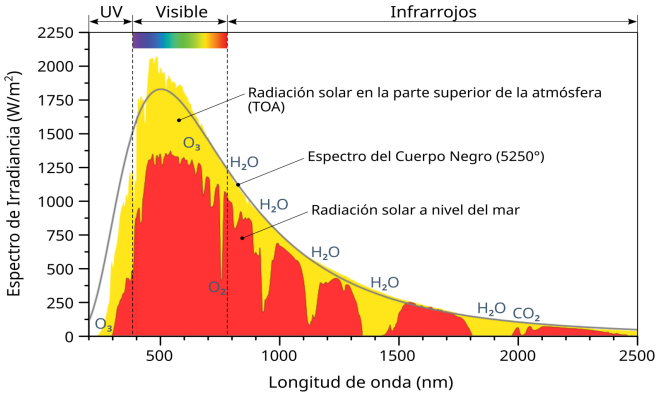


Figura 5.2 – *Espectro de irradiancia solar [8]*

Cuando se plantearon durante el diseño de la plataforma en el TFM se decidió apostar por la tecnología LED frente a otras alternativas como las lámparas **Arco de Xenón**, ya que los LEDs a pesar de no ofrecer una irradiancia tan completa como la alternativa, son mucho más económicos. En la Figura 5.3 se compara la irradiancia solar con la que ofrecen los simuladores solares de LED:

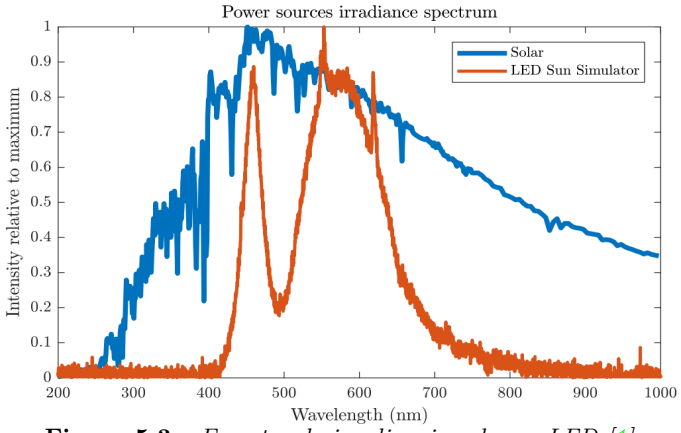


Figura 5.3 – *Espectro de irradiancia solar vs LED [1]*

Sin embargo, el principal desafío cuando se trabaja con LEDs es la disipación del calor que generan. En su día, estos se instalaron sobre un gran disipador. No obstante, con el fin de mejorar la disipación como se describe en el PM.Req.4, se diseñará además un sistema de **ventilación forzada**. El LED con el que trabajamos es un LED de potencia 14B10C cuyas especificaciones nos muestran que tiene una potencia máxima de 10 W y una corriente límite de 3 A.

El primer paso es caracterizar el dispositivo, esto es, obtener la **curva I-V** para posteriormente fijar una corriente de funcionamiento con un **driver** específico para iluminación LED. Para este fin se utiliza la fuente de alimentación del laboratorio y se construye la curva I-V que puede consultarse en la Figura 5.4:

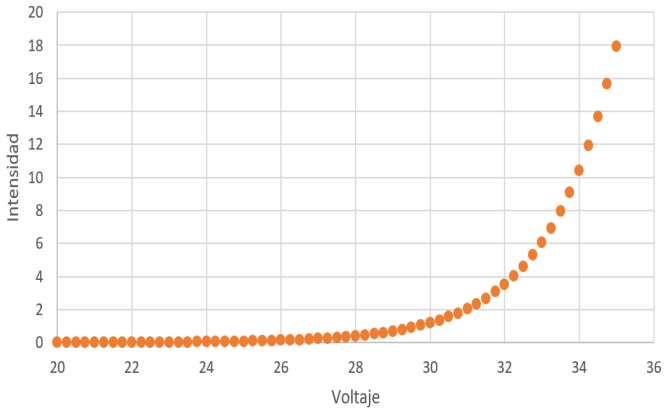


Figura 5.4 – *Curva I-V experimental del LED*

Se decide fijar la corriente dejando un margen del 10% del valor límite, lo que resulta en 2,7 A y 31,5 V. En paralelo con cada uno de los LEDs se plantea implementar un ventilador de 12 V que permita ayudar a la disipación de calor, para ello es necesario implementar una **resistencia de potencia** que actúe de **divisor de tensión**.

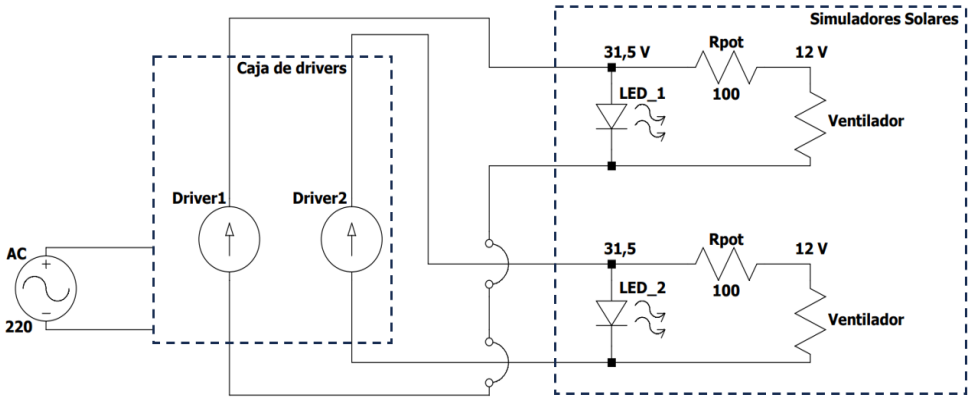


Figura 5.5 – *Circuito propuesto para los simuladores solares*

5.2 Estación Terrena

En lo referente a la estación terrena, en este capítulo ahondaremos en el funcionamiento de **OpenC3 COSMOS**, el ecosistema escogido para la comunicación con el CubeSat, definiremos las **interfaces de conexión**, y describiremos la **telemetría** y **telecomandos** que queremos mandar y recibir del satélite.

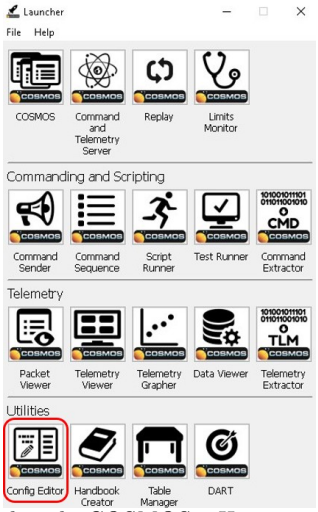


Figura 5.6 – *Launcher de COSMOS - Herramienta de configuración*

Usar COSMOS permite añadir una nueva capa de abstracción para desarrollar las herramientas necesarias para la misión, enfocándose en las funcionalidades requeridas en lugar de reescribir herramientas ya existentes, está escrito en **Ruby** y cuenta con una herramienta para configurar el ecosistema de forma fácil e intuitiva. La implementación de COSMOS que definirá nuestra estación terrena cuenta con la siguiente estructura de directorios:

```
COSMOS
├── Vendor
│   └── Ruby
└── CubeSat-I
    ├── lib
    ├── outputs
    ├── procedures
    ├── tools
    ├── CubeSat-I_Launcher.bat
    ├── Gemfile
    └── config
        ├── data
        ├── tools
        ├── system
        │   └── system.txt
        └── targets
            └── CubeSat-I
                ├── cmd_tlm
                │   ├── cmd.txt
                │   └── tlm.txt
                └── cmd_tlm_server.txt
```

5.2.1 Interfaz de conexión

El primer paso para configurar la estación terrena es definir las **interfaces de conexión** que necesitamos, para ello editamos el fichero:

./COSMOS/CubeSatI/config/targets/CubeSat-I/cmd_tlm_server.txt

Las interfaces pueden definirse de tal manera que COSMOS actúe como un **cliente** que se conecta al target o un **servidor** al que varios targets pueden conectarse. En cada caso, la estructura que sigue COSMOS para definir una interfaz es la siguiente:

Parámetro	Descripción	Obligatorio
Host	Máquina a la que conectarse	Sí
Puerto de escritura	Puerto de escritura de telecomandos	Sí
Puerto de lectura	Puerto de lectura de telemetría	Sí
Tiempo máx de escritura	Segundos que esperar antes de abortar la escritura	Sí
Tiempo máx de lectura	Segundos que esperar antes de abortar la lectura	Sí
Tipo de protocolo	Protocolo de comunicación	No

Tabla 5.1 – *Interfaz de cliente TCP/IP*

Parámetro	Descripción	Obligatorio
Puerto de escritura	Puerto de escritura de telecomandos	Sí
Puerto de lectura	Puerto de lectura de telemetría	Sí
Tiempo máx de escritura	Segundos que esperar antes de abortar la escritura	Sí
Tiempo máx de lectura	Segundos que esperar antes de abortar la lectura	Sí
Tipo de protocolo	Protocolo de comunicación	No

Tabla 5.2 – *Interfaz de Servidor TCP/IP*

A pesar de ser el protocolo TCP/IP el principal, COSMOS también soporta interfaces basadas en UDP y **Serial** con estructuras similares.

Para nuestro proyecto, acorde a lo definido en el requerimiento **ET.RF.1**, definiremos una interfaz cableada y otra inalámbrica, ambas basadas en un servidor TCP/IP al que el CubeSat podrá conectarse. Estableceremos en ambos casos un protocolo de comunicación **"Burst"** consistente en que COSMOS estará continuamente leyendo la telemetría recibida del satélite y mandará telecomandos cuando el usuario lo dicte.

5.2.2 Telemetría

La **telemetría** se envía desde el CubeSat como **paquetes de bytes** por lo que es necesario definir esos paquetes en COSMOS para que se interpreten correctamente. Para ello editamos el archivo:

./COSMOS/CubeSatI/config/targets/CubeSat.I/cmd_tlm/tlm.txt

En primer lugar, se define el **paquete** y luego se le añaden tantos **"items"** como se requiera, estos items son los diferentes **datos de telemetría** que se desean mandar en un mismo paquete, la estructura de los paquetes e items puede consultarse en las siguientes tablas:

Parámetro	Descripción	Obligatorio
Target	Nombre del satélite que manda la telemetría	Sí
Comando	Nombre del paquete de telemetría	Sí
Endianness	Formato en el que se reciben los datos	Sí
Descripción	Contenido del paquete de telemetría	No

Tabla 5.3 – *Definición de un paquete de Telemetría*

Parámetro	Descripción	Obligatorio
Nombre	Nombre del item de telemetría	Sí
Offset	Offset del bit más significativo del item	Sí
Tamaño	Tamaño en bits del item	Sí
Tipo de dato	int, uint, float, string, block o derived	Sí
Descripción	Descripción del item	No

Tabla 5.4 – *Definición de un Item de Telemetría*

Concretando de nuevo a nuestro satélite, definiremos un paquete de telemetría principal con un resumen de los datos más importantes sobre el estado del CubeSat y los valores de los sensores. Se definirá también un paquete de telemetría específico para consultar las medidas específicas directamente de cada sensor.

A modo de ejemplo, definiremos un paquete de telemetría para el sensor **MPU9250** donde se podrá consultar los valores individuales de aceleración y campo magnético en cada uno de los ejes X, Y y Z mientras que en el **panel del control** se podrá consultar la información referente a la orientación del satélite calculada a partir de dichos datos.

5

5.2.3 Telecomandos

Los **telecomandos** se envían desde la estación terrena con el fin de modificar el comportamiento de un satélite, en COSMOS se estructuran de forma muy similar a la telemetría, se define el comando y se le añade un item con la información que se quiere transmitir. Para definir un telecomando será necesario editar el archivo:

./COSMOS/CubeSatI/config/targets/CubeSat.I/cmd_tlm/cmd.txt

La estructura concreta de los telecomandos puede consultarse a continuación:

Parámetro	Descripción	Obligatorio
Target	Nombre del satélite objetivo	Sí
Comando	Nombre del paquete de telecomandos	Sí
Endianness	Formato en el que se envían los datos	Sí
Descripción	Contenido del paquete de telecomandos	No

Tabla 5.5 – *Definición de un paquete de Telecomandos*

Parámetro	Descripción	Obligatorio
Nombre	Nombre del item de telecomando	Sí
Offset	Offset del bit más significativo del item	Sí
Tamaño	Tamaño en bits del item	Sí
Tipo de dato	int, uint, float, string, derived o block	Sí
Parámetros	Valores máximo, minimo y por defecto	No

Tabla 5.6 – *Definición de un Item de Telecomando*

Los **telecomandos** necesarios para nuestro ejemplo de CubeSat serán los referentes al control de actitud. En concreto necesitamos indicarle al satélite cómo y cuánto debe activar el volante de inercia para reposicionarse. Se definirá por un lado un comando para activar el **modo de posicionamiento automático** en base a la orientación de la luz del sol y otros que permitan controlar **manualmente** la posición del satélite.

5.3 CubeSat

De nuevo nos centramos en el sistema clave de este proyecto, el satélite CubeSat. En esta sección trataremos en primer lugar de identificar los componentes y sistemas que necesiten ser reparados, sustituidos o directamente implementado de cero. Posteriormente definiremos el funcionamiento de los diferentes subsistemas referentes al OBC y el ADCS.

5.3.1 Mantenimiento y Reparación

Como se expuso en el primer capítulo, este proyecto parte de un prototipo del **GranaSat-I**, prototipo que quedó incompleto. Es por eso que ha sido necesario seguir un proceso de **ingeniería inversa**. Así, partiendo del esquemático del proyecto original se realiza una **inspección visual** mediante el microscopio del laboratorio y sucesivos **test de continuidad** en las diferentes pistas, consiguiendo entender bien la placa base del satélite e identificando los elementos que faltaban.

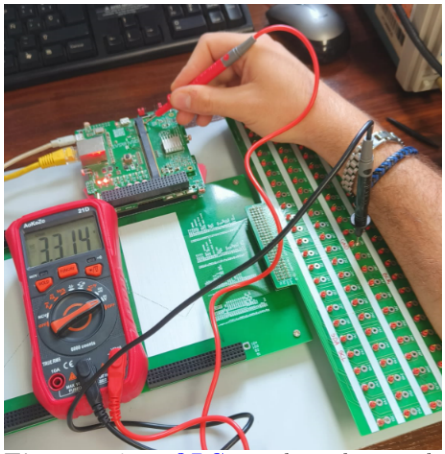


Figura 5.7 – *OBC en placa de test de continuidad de pistas*

Figura 5.8 – *Inspección de componentes en microscopio*

Como resultado se han identificado varios problemas en conexiones y componentes que no estaban implementados y que han dado lugar a la siguiente lista de tareas de mantenimiento:

Elemento	Problema detectado	Solución
RTC PCF8523	No instalado	Implementar
Resistencias LDR	No instaladas	Implementar
Sensor NYC70	Mal colocado	Reposicionar
Motor DC	Se desprendió por el gran par producido	Sustitución y diseño de soporte
Conexiones con ESP8266	Pistas no coinciden con esquemático	Corte de pistas y reconexión

Tabla 5.7 – *Listado de tareas de mantenimiento*

5.3.2 Ordenador de A Bordo

Pese a ser el sistema más importante del CubeSat, revisando la memoria del TFM encontramos poca documentación sobre el software implementado en él por lo que de nuevo es necesario seguir un proceso de **ingeniería inversa** para entender las herramientas que hay implementadas.

El primer paso es conectar la placa del OBC al ordenador del laboratorio para poder comunicarnos con la Raspberry CM3. Consultando de nuevo los esquemáticos, colocamos los **jumpers** en la posición correcta para poder alimentar la placa mediante un cable USB. Después, utilizaremos un **router**, al que conectaremos el ordenador del laboratorio y el OBC a través de **ethernet** y accediendo a la configuración LAN del mismo podremos asignar una ip fija al OBC y conectarnos a él a través de SSH mediante la herramienta MobaXterm.

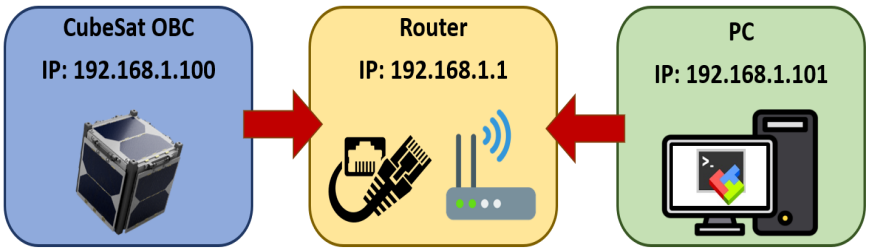


Figura 5.9 – *Conexión Ethernet al router del laboratorio*

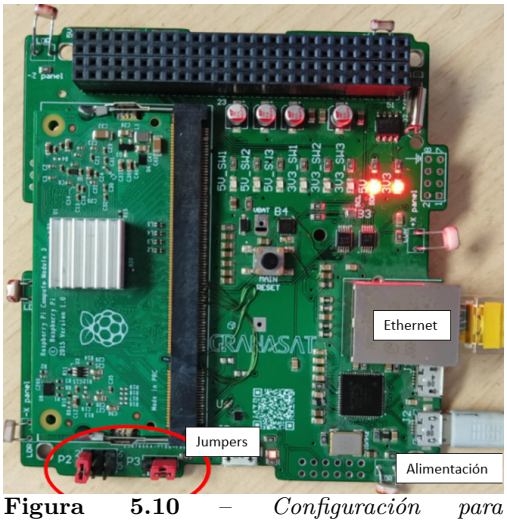


Figura 5.10 – *Configuración para alimentación y programación del OBC*

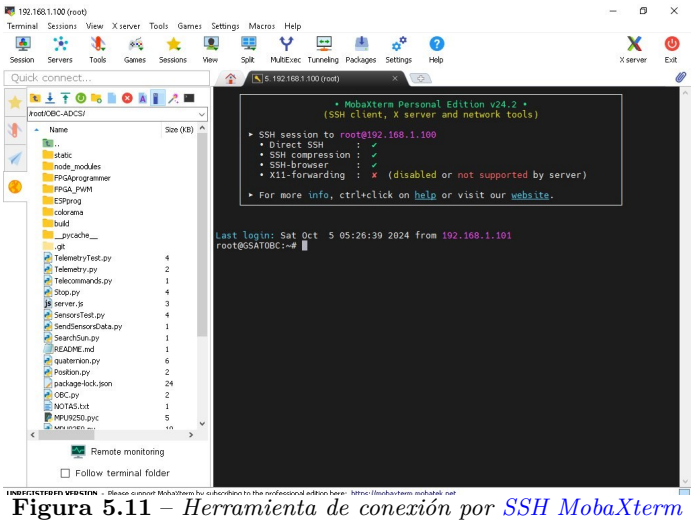


Figura 5.11 – *Herramienta de conexión por SSH MobaXterm*

Revisando los archivos presentes en la carpeta **/root/OBC-ADCS** identificamos diferentes archivos de código escritos en **python**, fundamentalmente librerías para los diferentes sensores y un código para probar la conexión por I2C con ellos. Tras haber estudiado los diferentes archivos y herramientas con las que se cuenta, podemos plantear la estructura funcional del código que controlará el comportamiento del CubeSat acorde a los requerimientos definidos, dicha estructura puede consultarse en la Figura 5.12.

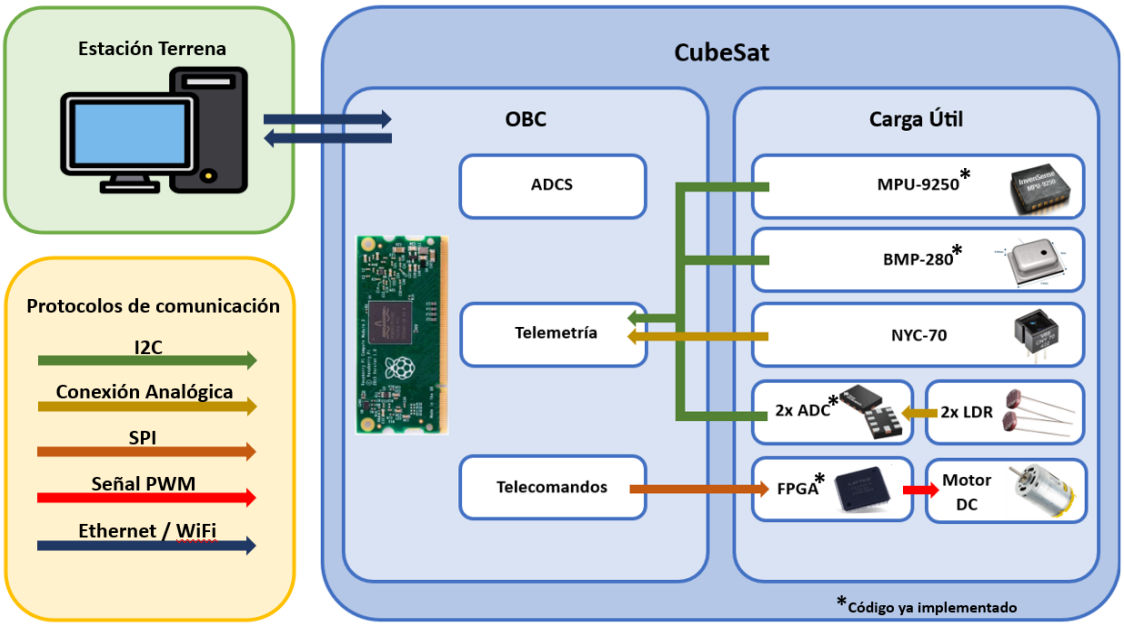


Figura 5.12 – *Estructura del software del OBC*

5.3.2.1 Sensores

En lo referente a los Sensores tras las implementación o sustitución de aquellos que fuera necesario, se estudia en profundidad las librerías para extraer datos de ellos y se comprueba el correcto funcionamiento de los mismos editando el archivo **/root/OBC-ADCS/SensorTest.py** para que se muestren los datos leídos por la **terminal**.

Cabe destacar que el **sensor CNY70**, responsable de medir la velocidad de giro del motor DC, no tiene librería implementada por lo que será necesario programarla.

```
MPU9250 SENSOR
IMU ACC : -0.0350 0.0740 -1.0290
IMU GYRO: 0.1600 º/s 1.1900 º/s -0.5490 º/s
IMU MAG : 7.5770 uT 30.7670 uT -95.9960 uT
IMU TEMP: 33.9870 ºC


LDR SENSORS
ADC0: CH1 86.0000 CH2 92.0000 CH3 264.0000 CH4 252.0000
ADC1: CH1 50.0000 CH2 13.0000 CH3 21.0000 CH4 68.0000


Bosh BPM280 SENSOR
BOSH TEMP : 31.2100 ºC
BOSH PRESS: 94065.4531 Pa
BOSH ALT  : 622.7200 m


NYC70 SENSOR
DC MOTOR VEL: 0.0000 rpm
```

Figura 5.13 – *Visualización de lectura de los sensores en la terminal*

5.3.2.2 Telemetría y Telecomandos

Atendiendo a la definición de paquetes de **telemetría** y **telecomandos** realizada en COSMOS será necesario convertir los diferentes valores extraídos de los sensores en el formato de **cadena de bits** que espera recibir la estación terrena. De igual manera, será necesario traducir e interpretar los telecomandos recibidos para que sea posible controlar el CubeSat.

5.3.3 Sistema de Determinación y Control de Actitud

Determinar y controlar la actitud de un satélite es crucial para que pueda cumplir con su misión, como apuntar sus cámaras o antenas en la dirección correcta.

Para determinar la actitud de nuestro CubeSat se calculará por un lado la **orientación** respecto al norte magnético terrestre programando una brújula a partir de los datos del **magnetómetro** incluido en el **sensor MPU9250**.

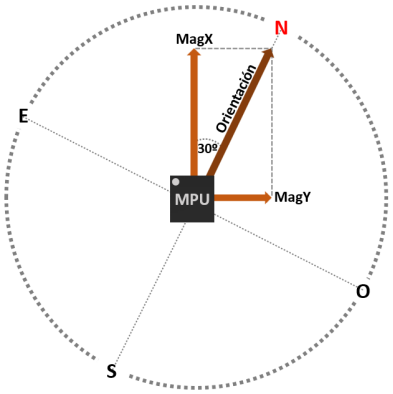


Figura 5.14 – *Cálculo de la orientación a partir de los valores del campo magnético*

$$\text{Orientación} = \arctan\left(\frac{\text{MagY}}{\text{MagX}}\right)$$

Por otro lado, también será necesario calcular la **posición respecto al sol**, para ello se hará uso de la orientación calculada anteriormente y los datos extraídos de los sensores LDR, que podremos interpretar de los ADCs.

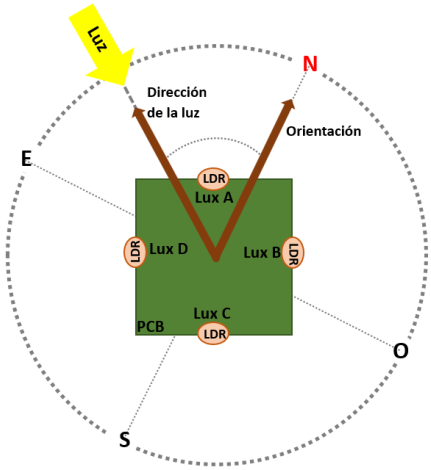


Figura 5.15 – *Cálculo de la dirección de la luz incidente*

$$x_{\text{vector}} = \text{luxA} - \text{luxC}$$
$$y_{\text{vector}} = \text{luxB} - \text{luxD}$$
$$\text{Dirección Luz} = \arctan\left(\frac{y_{\text{vector}}}{x_{\text{vector}}}\right)$$
$$\text{Posición Luz} = \text{Dirección Luz} - \text{orientación}$$

Una vez conocida la actitud del satélite y en función de los telecomandos recibidos, el CubeSat deberá poder **corregir su orientación** mediante el volante de inercia. Es aquí donde entra en juego la FPGA, que a partir de las instrucciones recibidas del OBC generará una señal PWM que permitirá controlar, a través de un puente H, la velocidad y el sentido de giro del motor DC.

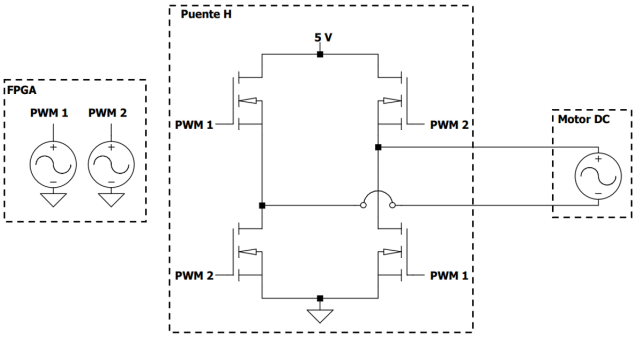


Figura 5.16 – *Control de Motor DC con Puente H*

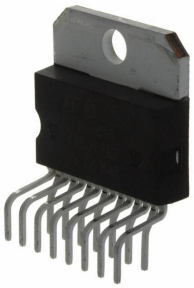


Figura 5.17 – *Puente H ST-L298N*

El código necesario para la generación de la señal PWM viene ya implementado, por lo que en lo referente a él nuestra tarea será **documentar su funcionamiento** y escribir el código que mandará las instrucciones para la generación de la señal desde el OBC.

5.3.3.1 Volante de Inercia

El volante de inercia está compuesto del motor DC descrito anteriormente sujeto a una carga pesada. Con ello se podrá generar el **par** suficiente al moverse para hacer girar el CubeSat cuando se encuentra en órbita sin rozamiento.

Originalmente el motor tenía las patillas soldadas directamente a la PCB, lo que provocó que la gran fuerza del par generado lo arrancara de la placa.

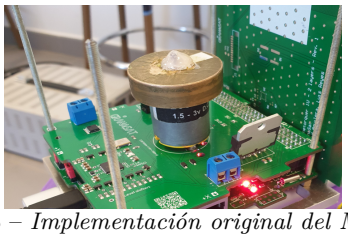


Figura 5.18 – *Implementación original del Motor DC [1]*

Es por eso que en lugar de soldarlo directamente a la PCB se diseñara, mediante la herramienta SolidWorks, un soporte que una el motor a la estructura del CubeSat, proporcionando una sujeción mucho más firme.

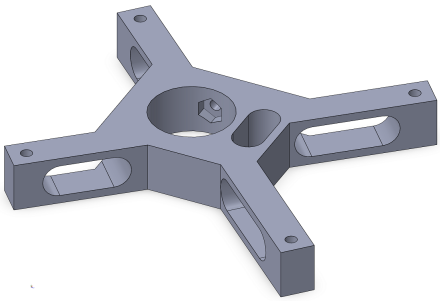


Figura 5.19 – *Visualización del soporte en SolidWorks*

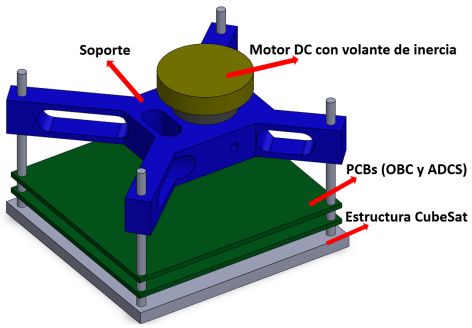


Figura 5.20 – *Render del CubeSat con el soporte*

Capítulo 6

Diseño del sistema

Una vez definidos los requerimientos técnicos y funcionales en la fase de **Ingeniería de Misión** y tras haber definido las soluciones a implementar en la fase de **Ingeniería de Sistemas** es el momento de abordar la última y principal etapa de ingeniería, el **diseño del sistema**.

Como en los capítulos anteriores, dividiremos esta sección en tres partes, para cada uno de los sistemas fundamentales del proyecto: la **plataforma mecánica**, la estación terrena y el **CubeSat**. Se detallará en cada sistema la implementación de las soluciones propuestas.

6.1 Plataforma Mecánica

6.1.1 I2DOS

La plataforma del I2DOS y la base han sido los sistemas que mejor se habían conservado, por lo que ha sido sufiente con una puesta a punto de las piezas antes de su montaje. Tras ajustar también el anillo inercial, el sistema queda listo para utilizarse.

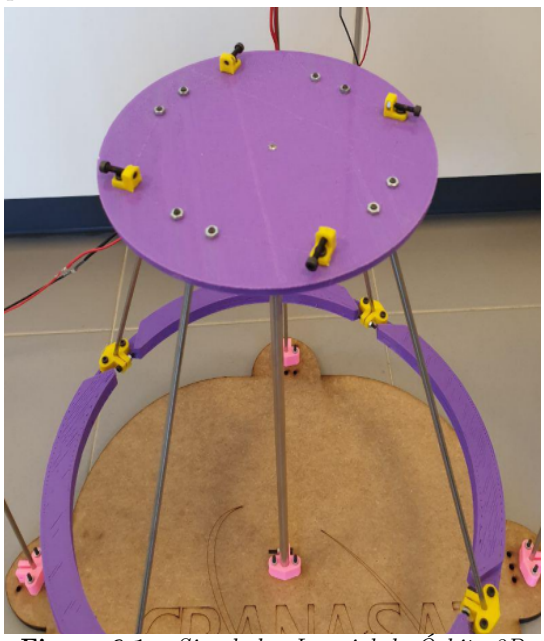


Figura 6.1 – *Simulador Inercial de Órbita 2D*

6.1.2 Simuladores Solares

Como se presentó en la sección 5.1.2, el LED propuesto para los simuladores solares es un LED 14B10C, que se monta sobre un **disipador** de calor de aluminio y se le añade un colimador para concentrar la luz en un haz lo más recto posible.

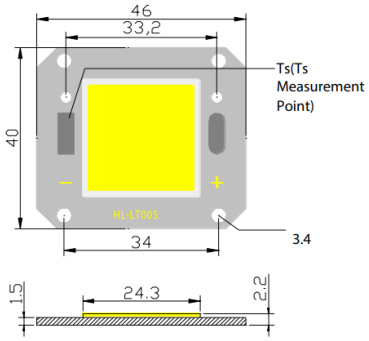


Figura 6.2 – *Dimensiones LED 14B10C [6]*

Figura 6.3 – *Simulador Solar Completo*

Para alimentar el circuito se implementan dos **drivers** de LED capaces de suministrar un voltaje de alrededor de 32 V con una potencia máxima de 100 W. Alimentar un LED con un driver especializado para ese fin, permite que se trabaje en un punto de operación más seguro, ya que el driver se asegura de mantener constante la corriente en vez del voltaje. Controlar un LED por voltaje es arriesgado puesto que pequeñas variaciones en la tensión se traducen en grandes variaciones de corriente que pueden dañar el dispositivo.

Figura 6.4 – *Drivers elegidos para el control de los LEDs*

Con el fin de implementar el circuito planteado en la figura 3.4 para la alimentación del ventilador, se utilizará cable de sección 0,50 mm^2, capaz de soportar hasta 5 A. Para la **resistencia de potencia** se elige una de valor 100 Ω y 5 W.

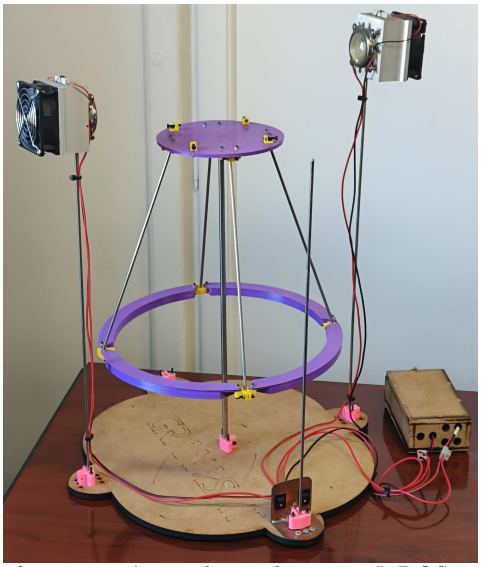


Figura 6.5 – *Plataforma mecánica al completo, con I2DOS y simuladores solares*

6.2 Estación Terrena

Esta sección se centrará en la programación de la estación terrena de acuerdo con la estructura y las definiciones de datos expuestas en el capítulo anterior.

6.2.1 Interfaz de conexión

Se define un servidor TCP/IP llamado CUBESAT1_INT que utilizará el mismo puerto para la lectura y la escritura. Establecemos el protocolo de conexión **Burst** que consiste en leer tantos datos de la interfaz como haya disponibles.

```
1 INTERFACE CUBESAT1_INT tcpip_server_interface.rb  8445 8445 10.0 nil BURST
2  TARGET CUBESAT_1
```

Listado 6.1 – *Definición de interfaz de conexión*

Una vez definido podemos lanzar el servidor desde el **launcher de COSMOS**.

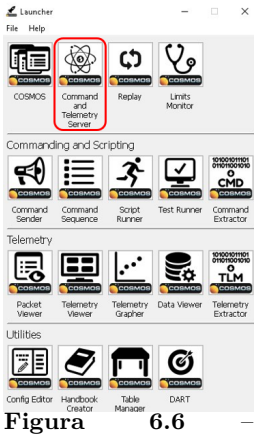


Figura 6.6 – *Launcher de COSMOS (cmd_tlm_server)*

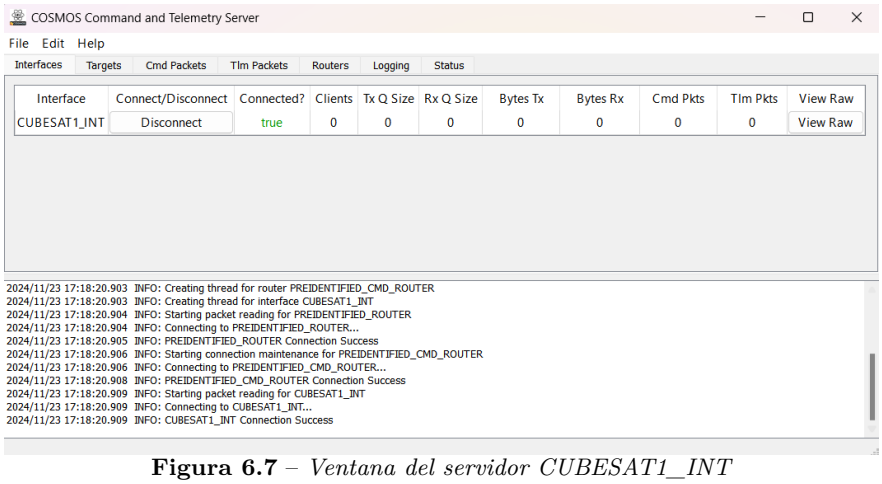


Figura 6.7 – *Ventana del servidor CUBESAT1_INT*

6

6.2.2 Telemetría

Para la telemetría se definen cinco paquetes, uno para cada uno de los sensores y un quinto llamado **Panel de Control** que reúna la información más relevante de los diferentes sensores y muestre los cálculos de actitud. Cada paquete tendrá un listado de **items** asignado en función de los valores que se esperen recibir de cada sensor.

```
1  TELEMETRY GRANASAT_1 PANEL_DE_CONTROL BIG_ENDIAN "Readings of the diferent
     payload sensors"
2    # Keyword       Name              BitSize Type   ID Description
3    APPEND_ID_ITEM  ID                16      INT    1  "Identifier"
4    APPEND_ITEM     Temperature       32      FLOAT     "CubeSat Temperature"
5    APPEND_ITEM     Altitude          32      FLOAT     "CubeSat Altitude"
6    APPEND_ITEM     Pression          32      FLOAT     "CubeSat Pression"
7    APPEND_ITEM     Orientation       32      FLOAT     "CubeSat Orientation"
8    APPEND_ITEM     Sun_direction     32      FLOAT     "Lighth direction"
9    APPEND_ITEM     FlyWheel_Vel      32      FLOAT     "FlyWheel velocity"
10
11 TELEMETRY GRANASAT_1 SENSOR_MPU BIG_ENDIAN "Readings of MPU9250 sensor"
```

```
# Keyword          Name       BitSize  Type    ID  Description
APPEND_ID_ITEM     ID         16       INT     2   "Identifier"
APPEND_ITEM        Gyro_X     32       FLOAT       "Gyro X readings"
APPEND_ITEM        Gyro_Y     32       FLOAT       "Gyro Y readings"
APPEND_ITEM        Gyro_Z     32       FLOAT       "Gyro Z readings"
APPEND_ITEM        Acc_X      32       FLOAT       "Acc X readings (position)"
APPEND_ITEM        Acc_Y      32       FLOAT       "Acc Y readings (position)"
APPEND_ITEM        Acc_Z      32       FLOAT       "Acc Z readings (position)"
APPEND_ITEM        Mag_X      32       FLOAT       "Mag X readings (uT)"
APPEND_ITEM        Mag_Y      32       FLOAT       "Mag Y readings (uT)"
APPEND_ITEM        Mag_Z      32       FLOAT       "Mag Z readings (uT)"
APPEND_ITEM        TempMPU    32       FLOAT       "MPU Temp readings"

TELEMETRY GRANASAT_1 ADC BIG_ENDIAN "Readings of ADS1015 converter"
# Keyword          Name       BitSize  Type    ID  Description
APPEND_ID_ITEM     ID         16       INT     3   "Identifier"
APPEND_ITEM        ADC0_CH1   32       FLOAT       "LDR +Y"
APPEND_ITEM        ADC0_CH2   32       FLOAT       "LDR +X"
APPEND_ITEM        ADC0_CH3   32       FLOAT       "Sense A Motor"
APPEND_ITEM        ADC0_CH4   32       FLOAT       "Sense B Motor"
APPEND_ITEM        ADC1_CH1   32       FLOAT       "LDR -Y"
APPEND_ITEM        ADC2_CH2   32       FLOAT       "LDR -Z"
APPEND_ITEM        ADC3_CH3   32       FLOAT       "LDR +Z"
APPEND_ITEM        ADC4_CH4   32       FLOAT       "LDR -X"

TELEMETRY GRANASAT_1 BOSH BIG_ENDIAN "Readings of BMP280 sensor"
# Keyword          Name       BitSize  Type    ID  Description
APPEND_ID_ITEM     ID         16       INT     4   "Identifier"
APPEND_ITEM        TempBOSH   32       FLOAT       "BOSH Temp readings"
APPEND_ITEM        Press      32       FLOAT       "Pression readings"

TELEMETRY GRANASAT_1 NYC70 BIG_ENDIAN "Readings of NYC70 sensor"
# Keyword          Name       BitSize  Type    ID  Description
APPEND_ID_ITEM     ID         16       INT     5   "Identifier"
APPEND_ITEM        FlyWheel_Vel 32     FLOAT       "FlyWheel velocity"
```

Listado 6.2 – *Definición de Telemetría*

Para visualizar los paquetes de telemetría tenemos varias opciones en el launcher. A continuación, se muestra una imagen de la función **Packet Viewer**.

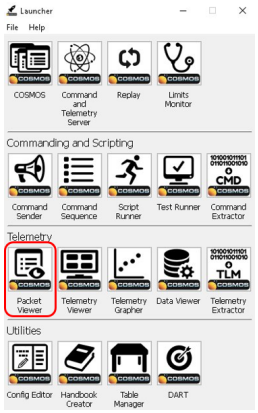


Figura 6.8 – *Launcher de COSMOS (Packet Viewer)*

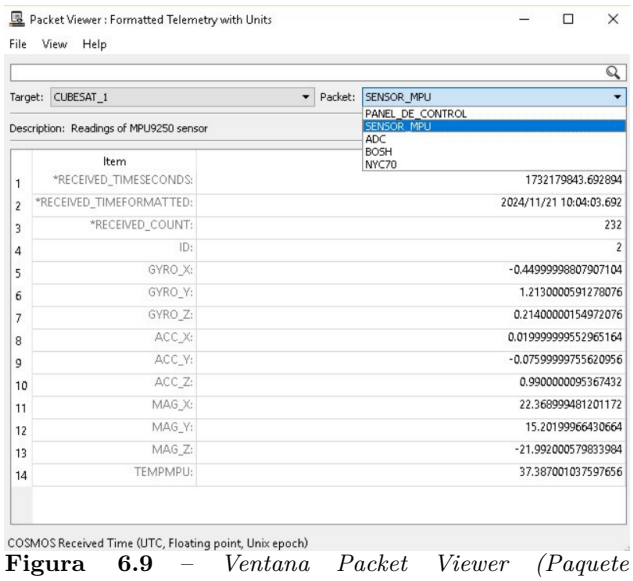


Figura 6.9 – *Ventana Packet Viewer (Paquete SENSOR_MPU)*

6.2.3 Telecomandos

Por último, para definir los **telecomandos** crearemos un paquete diferente para cada una de las instrucciones que queremos mandar. Cada paquete tendrá solo un **item**, aparte del identificador, ya que las órdenes serán siempre del tipo ON/OFF.

```
COMMAND GranaSat_1 ADCS BIG_ENDIAN "Command to order the CubeSat to position
  himself towards de sun"
  # Keyword              Name          BitSize  Type   Min Max  Def  Description
  APPEND_ID_PARAMETER ID              16       INT    1   1    1    "Identifier"
  APPEND_PARAMETER    Search_Sun      8        UINT   MIN MAX  0    "ON/OFF"
    STATE FALSE 0
    STATE TRUE 1

COMMAND GranaSat_1 Manual_Right BIG_ENDIAN "Command to turn CubeSat right"
  # Keyword              Name          BitSize  Type   Min Max  Def  Description
  APPEND_ID_PARAMETER ID              16       INT    2   2    2    "Identifier"
  APPEND_PARAMETER    Manual_Right    8        UINT   MIN MAX  0    "Turn Right"
    STATE FALSE 0
    STATE TRUE 1

COMMAND GranaSat_1 Manual_Left BIG_ENDIAN "Command to turn CubeSat left"
  # Keyword              Name          BitSize  Type   Min Max  Def  Description
  APPEND_ID_PARAMETER ID              16       INT    3   3    3    "Identifier"
  APPEND_PARAMETER    Manual_Left     8        UINT   MIN MAX  0    "Turn Left"
    STATE FALSE 0
    STATE TRUE 1
```

Listado 6.3 – *Definición de Telecomandos*

De nuevo accedemos a los telecomandos desde el launcher mediante la función **Command Sender**.

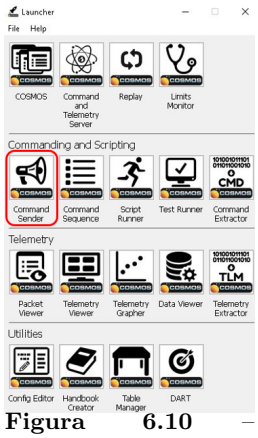


Figura 6.10 – *Launcher de COSMOS (Packet Viewer)*

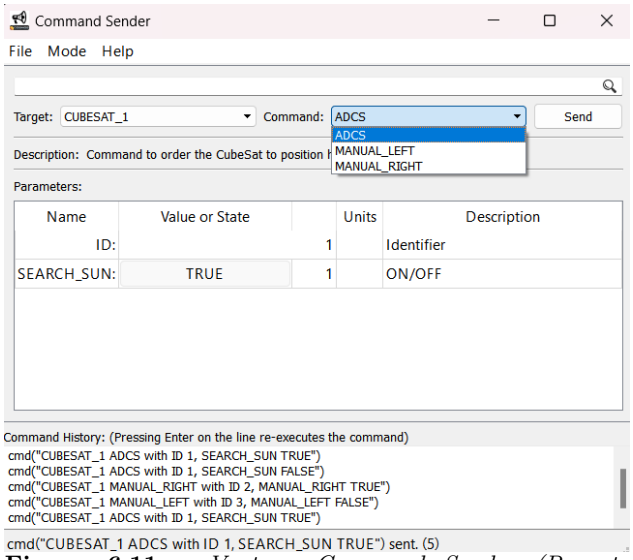


Figura 6.11 – *Ventana Command Sender (Paquete ADCS)*

6.3 CubeSat

6.3.1 Mantenimiento y reparación

Lo primero que se aborda respecto al CubeSat es el listado de tareas de mantenimiento (Tabla 5.7). Se implementa el RTC y las seis resistencias LDR, se recoloca el **sensor NYC70** aprovechando el nuevo soporte del motor y se corrige las conexiones con la **ESP8266**.

Este último problema, al ser un defecto de diseño en lugar de un problema derivado del deterioro, requeriría el rediseño de una nueva **PCB** y volver a implementar todos los componentes, lo que resulta en una tarea tan extensa y complicada que podría abarcar un TFG aparte. En concreto las conexiones que son erróneas son aquellas que llegan a los pines **TX** y **RX**, necesarias para la comunicación con la **Raspberry CM3**. Como solución se decide cortar manualmente las pistas y realizar un puente con un cable especial para este fin.

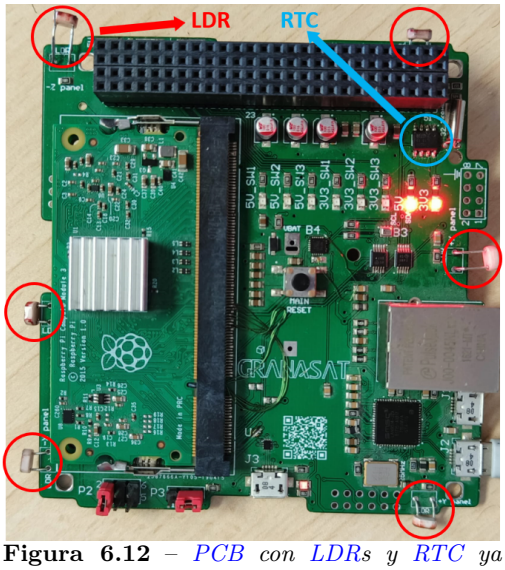


Figura 6.12 – *PCB con LDRs y RTC ya soldados*

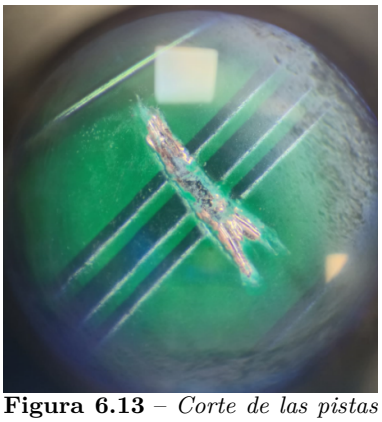


Figura 6.13 – *Corte de las pistas visto a través del microscopio*

6.3.2 Conexión con Estación Terrena

Para establecer un canal de comunicación eficaz con la estación terrena es necesario que el CubeSat se conecte al **servidor TCP/IP** que se ha definido en COSMOS, para ello en el código del programa principal se incluyen las siguientes sentencias:

```
#Connect to COSMOS server
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = '192.168.1.101'
port = 8445
s.connect((host, port))
s.setblocking(0)
```

Listado 6.4 – *Conexión al servidor COSMOS*

6.3.2.1 Envío de telemetría

Para el envío de **telemetría** se implementa una función que se llame desde el programa principal y se encargue de traducir los valores de las magnitudes recibidas de los sensores al formato **Big Endian** que espera recibir COSMOS.

```
import socket
import sys
import time
import struct
import select
import numpy as np
from MPU9250 import MPU9250
from ADS1015 import ADS1015
from BMP280 import BMP280,BMP280_ULTRAHIGHRES
from CNY70 import CNY70

class Telemetry:
  def SendTLM():
    #definicion de sensores a utilizar
    imu = MPU9250()
    adc0 = ADS1015(address=0x48)
    adc1 = ADS1015(address=0x49)
    bosh = BMP280(mode=BMP280_ULTRAHIGHRES)
    cny70 = CNY70()

    #Funcion para el calculo de la orientacion de grados con el magnetometro
    def brujula(magX, magY):
      heading_rd = np.arctan2(magY, magX)
      heading = np.degrees(heading_rd)
      if heading <0:
        heading += 360
      return heading

    #Funcion para calcular de donde llega la luz
    def calc_ubi_sol(lux1, lux2, lux3, lux4, heading):
      x_vector = lux1 - lux3
      y_vector = lux2 - lux4
      angle = np.arctan2(y_vector, x_vector)*(180/ np.pi)
      angle += 90
      if heading <0:
        heading += 360
      ubi_sol = (angle + heading) % 360
      return ubi_sol

    #RADING SENSORS
    # IMU
    tempimu = imu.readTemperature()
    gyro = imu.readGyro()
    acc = imu.readAccel()
    mag = imu.readMagnet()
    orientation = brujula(mag['x'], mag['y'])
    #ADC
    adc0read = [0]*4
    adc1read = [0]*4
    for i in range(4):
      adc0read[i] = adc0.read_adc(i, gain=1)
      adc1read[i] = adc1.read_adc(i, gain=1)

    sun_direction = calc_ubi_sol(adc0read[1], adc0read[0], adc1read[3], adc1read
  [0], orientation)

    # BOSH
    tempbosh = bosh.read_temperature()
    press = bosh.read_pressure()
    altitude = bosh.read_altitude()

    temp = (tempimu + tempbosh)/2

    # NYC70
    flywheel_vel = cny70.medirRPM()
    # GENERATING TELEMETRY
```

```
66      TLM_packet_1 = struct.pack('>Hffffff', 1, temp, altitude, press, orientation
        , sun_direction, flywheel_vel)
67
68      TLM_packet_2 = struct.pack('>Hfffffffffff', 2, gyro['x'], gyro['y'], gyro['z'
        ], acc['x'], acc['y'], acc['z'], mag['x'], mag['y'], mag['z'], tempimu)
69
70      TLM_packet_3 = struct.pack('>Hfffffffff', 3, adc0read[0], adc0read[1],
        adc0read[2], adc0read[3], adc1read[0], adc1read[1], adc1read[2], adc1read[3])
71
72      TLM_packet_4 = struct.pack('>Hff', 4, tempbosh, press)
73
74      TLM_packet_5 = struct.pack('>Hf', 5, flywheel_vel)
75
76
77      return TLM_packet_1, TLM_packet_2, TLM_packet_3, TLM_packet_4, TLM_packet_5
```

Listado 6.5 – *código Telemetry.py*

6.3.2.2 Recepción de telecomandos

Simultáneamente al envío de telemetría el CubeSat debe estar a la espera de recibir telecomandos, que se mandan también en formato **Big Endian**, y debe poder interpretarlos y actuar en consecuencia.

```
1  import socket
2  import sys
3  import time
4  import struct
5  import select
6  import numpy as np
7  from GiroMotor import motorDC
8
9  class Telecommands:
10   #Funcion para interpretar los comandos recibidos
11   command_id = 0
12   command = 0
13
14   def readCMD(data):
15     command_id, command = struct.unpack('>HB',data)
16     print("TeleComando recibido")
17
18     if command_id == 1:  #Modo Search_sun
19       if command == 1:
20         return 1
21       else:
22         motorDC.parar()
23         return 0
24
25     elif command_id == 2:  #Giro manual a la izquierda
26       if command == 1:
27         motorDC.giroIzq(255)
28         return 0
29       else:
30         motorDC.giroIzq(0)
31         return 0
32
33     elif command_id == 3:    #Giro manual a la derecha
34       if command == 1:
35         motorDC.giroDer(255)
36         return 0
37       else:
38         motorDC.giroDer(0)
39         return 0
```

Listado 6.6 – *Código Telecommands.py*

6.3.3 Sistema de determinación y control de actitud

6.3.3.1 Soporte mecánico del volante de inercia

Debido a las estrictas restricciones de peso presentes en cualquier aplicación espacial el material del soporte diseñado para el volante de inercia se decide fabricar mediante **impresión 3D**. En la Figura 6.14 puede verse el resultado, con el volante de inercia ya implementado.

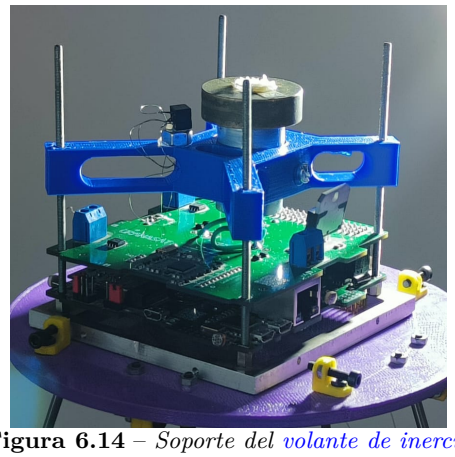


Figura 6.14 – *Soporte del volante de inercia*

6.3.3.2 Algoritmos de determinación de actitud

Se aborda a continuación la implementación de los algoritmos de determinación de actitud descritos en la sección 5.3.3. Ambos se definirán como funciones que devuelvan los valores de **Orientación** y **Posición de Luz** respectivamente a partir de los valores de los sensores introducidos.

```
import time
import numpy as np
from GiroMotor import motorDC

def orient_solar_panel(sun_angle, panel_angle):
  # Calcula la diferencia angular
  diff_angle = (sun_angle - panel_angle) % 360
  # Si la diferencia es mayor de 180, girar en la direccin opuesta
  if diff_angle > 180:
    diff_angle -= 360
  return diff_angle

class SearchSun:
  def searchsun(sun_direction, orientation):
    diff_angle = orient_solar_panel(sun_direction, orientation)
    if -10 <= abs(diff_angle) <= 10:
      motorDC.parar()
      print(diff_angle)
      print("La placa esta alineada con el sol")
      return 1
    elif diff_angle < 0:
      motorDC.giroIzq(255)
      print(diff_angle)
      print("Girando a la izquierda")
      return 0
    else:
      motorDC.giroDer(255)
      print(diff_angle)
      print("Girando a la derecha")
      return 0
```

Listado 6.7 – *Conexión al servidor COSMOS*

6.3.3.3 Control del volante de inercia

Para el manejo del volante de inercia es necesario definir una librería que contenga diferentes funciones para controlar la puesta en marcha, velocidad y paro del motor DC. Esta librería escribirá las órdenes pertinentes a través de SPI a la FPGA, que generará la señal PWM necesaria para modificar el comportamiento del motor DC.

```
import spidev
import time
import RPi.GPIO as gpio

gpio.setmode(gpio.BCM)
gpio.setup(7, gpio.OUT)
gpio.output(7, True)
gpio.setup(44, gpio.OUT)
gpio.output(44, False)
time.sleep(0.1)
gpio.output(44, True)

spi = spidev.SpiDev()
spi.open(0, 0)
spi.max_speed_hz = 5000000
def writePWM(ch,val):
    gpio.output(7, False)
    spi.xfer2([ch,val])
    gpio.output(7, True)
writePWM(0,10)

for i in [1,2,3,4,5,6,7,8,9]:
    writePWM(i,0)
class motorDC:
  def giroDer(vel):   #Velocidad entre 0 y 255
    writePWM(2,vel)
    writePWM(1,0)
  def giroIzq(vel):   #Velocidad entre 0 y 255
    writePWM(2,0)
    writePWM(1,vel)
  def parar():
    writePWM(1,0)
    writePWM(2,0)
```

Listado 6.8 – *Conexión al servidor COSMOS*

6.3.4 Programa principal del OBC

El **programa principal** hace referencia al algoritmo que se ejecutará en **bucle** mientras el CubeSat está en operación. Dicho algoritmo es el encargado de llamar a las diferentes funciones descritas en los apartados anteriores. Cuando se ejecuta, realiza primero varias tareas de inicialización, tales como cargar el programa en la FPGA, calibrar los sensores y conectarse a la estación terrena. Después entra en un bucle en el que manda los datos de telemetría en tiempo real, interpreta los telecomandos recibidos y dirige las funciones de control de actitud.

```
import socket
import sys
import time
import select
import subprocess
import struct
from Telemetry import Telemetry
from Telecommands import Telecommands
from MPU9250 import MPU9250
from ADS1015 import ADS1015
from BMP280 import BMP280,BMP280_ULTRAHIGHRES
```

```
12 from CNY70 import CNY70
13 from GiroMotor import motorDC
14 from SearchSun import SearchSun
15
16 flag1 = 0
17 flag2 = 0
18 #Connect to COSMOS server
19 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
20 host = '192.168.1.101'
21 port = 8445
22 s.connect((host, port))
23 s.setblocking(0)
24
25 #Programar FPGA
26 fpga_prog = "sudo /root/OBC-ADCS/FPGAprogrammer/fpgaprog /root/OBC-ADCS/FPGA_PWM
    /main.bin"
27 subprocess.run(fpga_prog, shell=True, check=True)
28
29 #Calibrar MPU
30 imu = MPU9250()
31 imu.calibrate()
32
33 mean_offset_x=0
34 mean_offset_y=0
35 mean_offset_z=0
36
37 for i in range(1,5000):
38     data_loop=imu.readGyro()
39     data = ([x for x in [data_loop['x'],data_loop['y'],data_loop['z']]])
40     mean_offset_x=(mean_offset_x+data[0])
41     mean_offset_y=(mean_offset_y+data[1])
42     mean_offset_z=(mean_offset_z+data[2])
43
44 mean_offset_x=mean_offset_x/i
45 mean_offset_y=mean_offset_y/i
46 mean_offset_z=mean_offset_z/i
47
48 #Bucle
49 while True:
50   #Send Telemetry
51   TLM1, TLM2, TLM3, TLM4, TLM5 = Telemetry.SendTLM()
52   s.sendall(TLM1)
53   time.sleep(0.05)
54   s.sendall(TLM2)
55   time.sleep(0.05)
56   s.sendall(TLM3)
57   time.sleep(0.05)
58   s.sendall(TLM4)
59   time.sleep(0.05)
60   s.sendall(TLM5)
61   #Read Telecommands
62   readable, _, _ = select.select([s], [], [], 0.01)
63   if readable:
64     data = s.recv(3)
65     flag1 = Telecommands.readCMD(data)
66
67   #Funcion Search Sun
68   _, _, _, _, orientation, sun_direction, _, = struct.unpack('>Hffffff', TLM1)
69   if flag1 == 1:
70     flag2 = SearchSun.searchsun(sun_direction, orientation)
71     print(sun_direction)
72     if flag2 == 1:
73       flag2=0
74   time.sleep(0.5)
```

Listado 6.9 – *Código OBC.py*

Capítulo 7

Validación

En este quinto capítulo se validan las soluciones propuestas en este Trabajo Fin de Grado y comprobar el cumplimiento de los requerimientos definidos. Esta verificación se lleva a cabo utilizando dos recursos: por un lado, diferentes **fotografías y videos de las soluciones desarrolladas**, que complementan las introducidas en cada sección del capítulo 6 y por otro lado diferentes **test** para comprobar el cumplimiento de los diferentes requisitos técnicos.

7.1 Plataforma mecánica

En lo referente a la plataforma mecánica podemos afirmar que se han cumplido correctamente diferentes requerimientos técnicos y funcionales. Tanto el I2DOS como los simuladores solares funcionan correctamente. A continuación, se muestran dos imágenes de la plataforma mecánica al completo con el CubeSat listo para iniciar los test pertinentes.

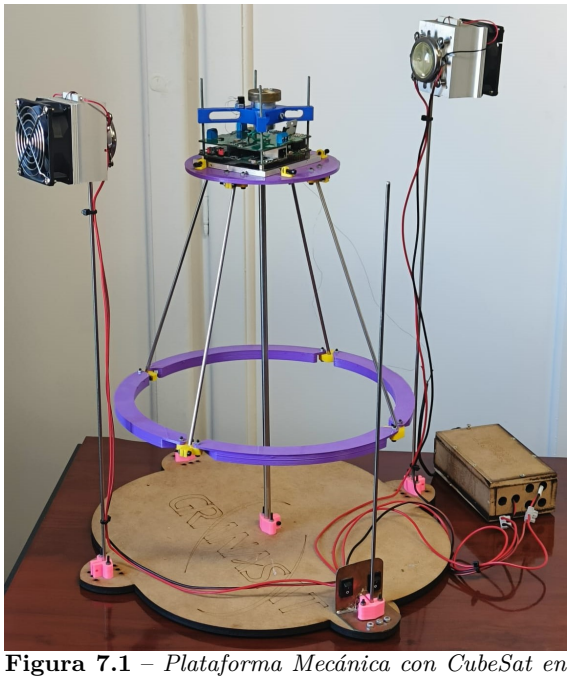


Figura 7.1 – *Plataforma Mecánica con CubeSat en posición*

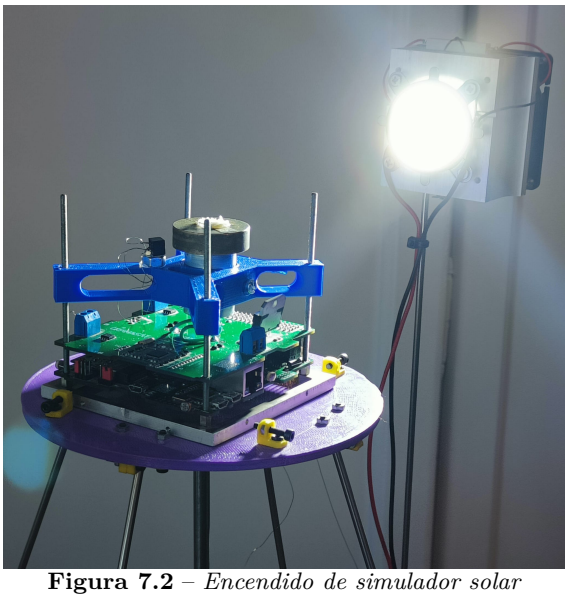


Figura 7.2 – *Encendido de simulador solar*

Centrándonos en el requerimiento **PM.Req.4** sobre la correcta refrigeración de los simuladores solares y con el fin de estudiar si es efectiva la implementación del ventilador se deja encendido uno de los LEDs durante una hora y se mide la temperatura tanto en el propio LED como en la **resistencia de potencia**. Para ello, se utiliza una cámara termográfica Fluke Ti105 comprobándose que ambos se mantienen en un rango seguro de operación.

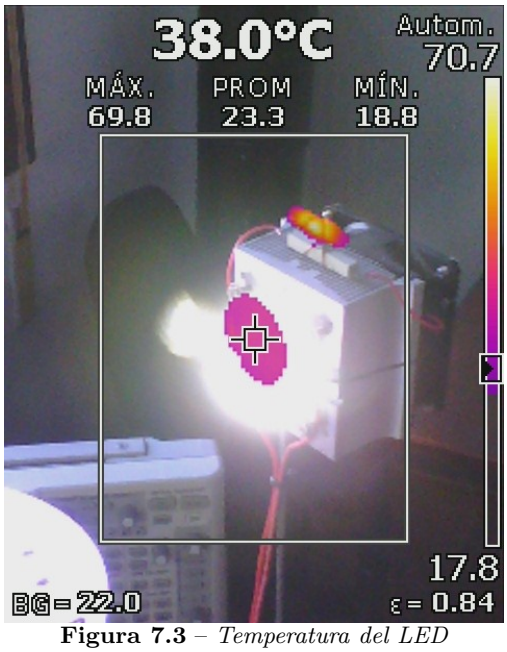


Figura 7.3 – *Temperatura del LED*

Figura 7.4 – *Temperatura de la Resistencia*

7.2 Estación Terrena y CubeSat

La validación de la estación terrena y el CubeSat se realizan en la misma sección ya que son sistemas codependientes, de esta manera se verificará el correcto funcionamiento de los diferentes sensores recibiendo y graficando sus lecturas en COSMOS y se comprobará el correcto funcionamiento del ADCS del satélite mandando los telecomandos requeridos.

7.2.1 Telemetría

Para verificar la recepción de telemetría y, por ende, el correcto funcionamiento de los sensores del satélite utilizaremos la función **Telemetry Grapher** de COSMOS. En la Figura 7.5 se muestran las lecturas de presión, temperatura y altitud del sensor **BOSH BMP28** y las lecturas de campo magnético del sensor **MPU9250**.

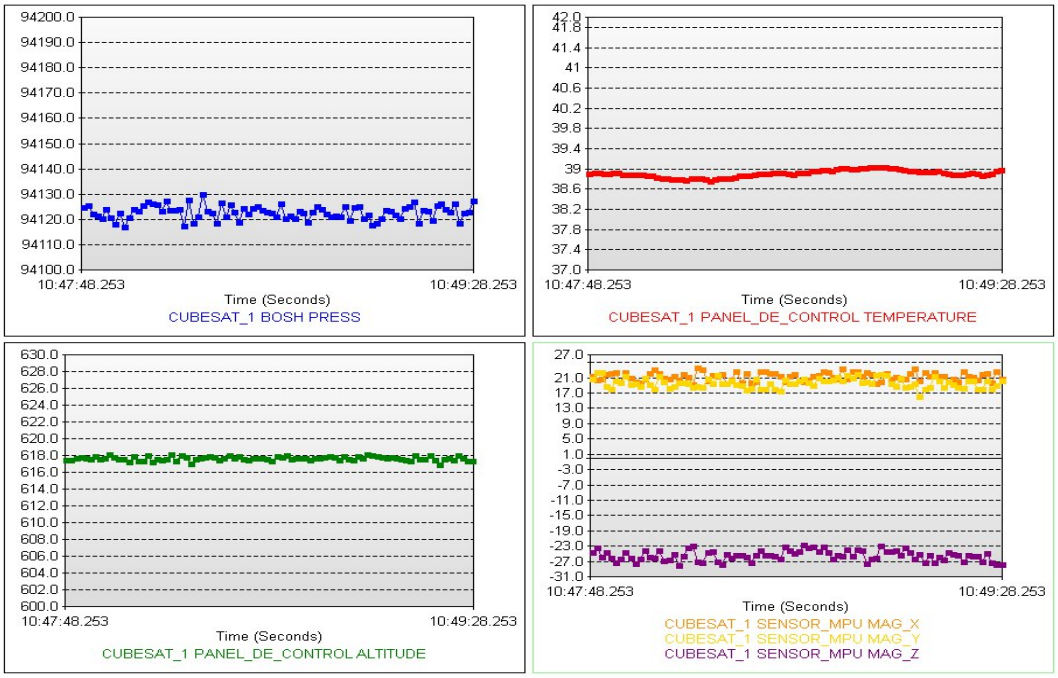


Figura 7.5 – *Lectura de sensores en tiempo real en COSMOS*

7.2.2 Telecomandos y ADCS

En este segundo test se ordenará al CubeSat orientarse de forma autónoma hacia la luz del simulador solar encendido mediante el telecomando programado en COSMOS **"Search_Sun"**. Se graficarán simultáneamente las medidas de **orientación** y **dirección del sol** junto a la **velocidad del volante de inercia**. Una vez el CubeSat esté orientado correctamente se comprobará como responde a las perturbaciones en su orientación.

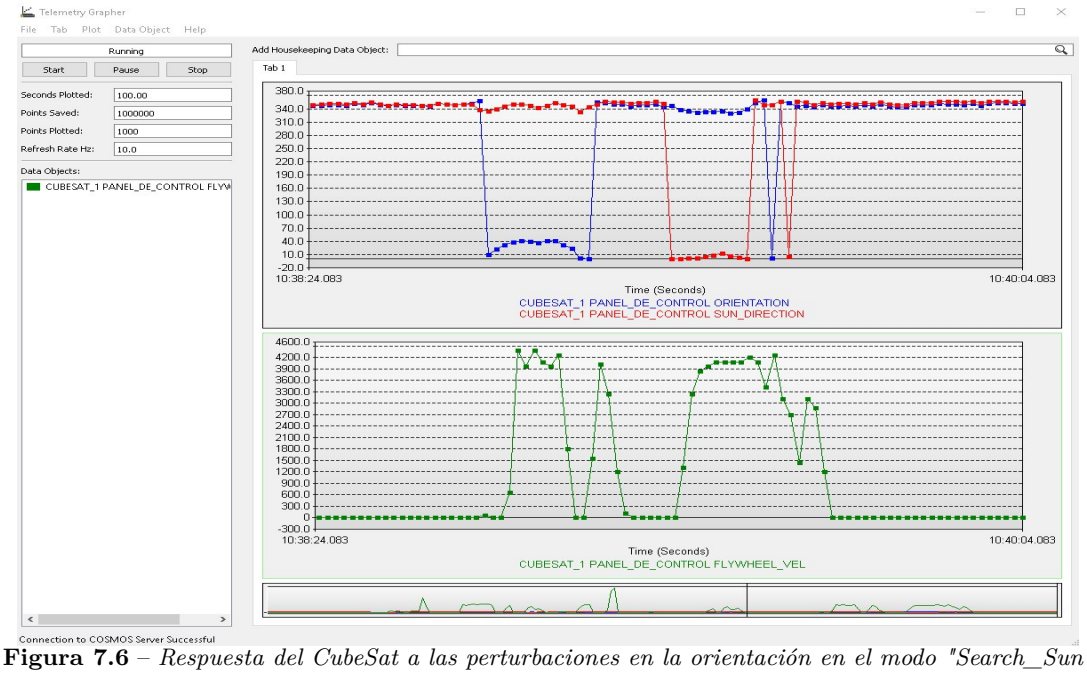
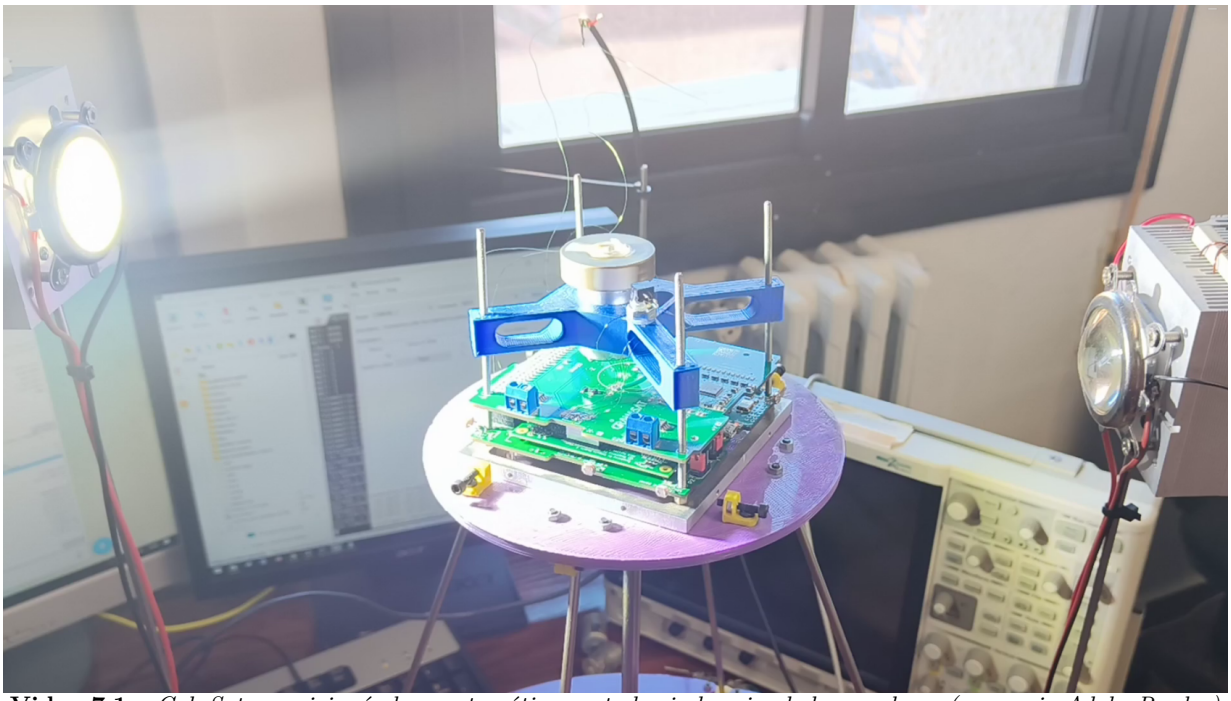


Figura 7.6 – *Respuesta del CubeSat a las perturbaciones en la orientación en el modo "Search_Sun"*

Se incluye también un video mostrando la respuesta a perturbaciones y cambios en la dirección de la luz cuando el modo **"Search_Sun"** esta activado:



Video 7.1 – *CubeSat reposicionándose automáticamente hacia los simuladores solares (necesario Adobe Reader)*

7.2.3 Consumo de potencia del CubeSat en operación

Un reducido **Consumo de potencia** es crucial en cualquier aplicación espacial, en nuestro caso el requerimiento **CS.Req.9** marcaba el consumo de máximo del CubeSat en operación en 5 W.

Para realizar el estudio del consumo se utilizará el Power Analyzer N6705A de **Agilent Technologies** que se encuentra en el laboratorio. Los resultados obtenidos por el mismo pueden consultarse en la Figura 7.8.

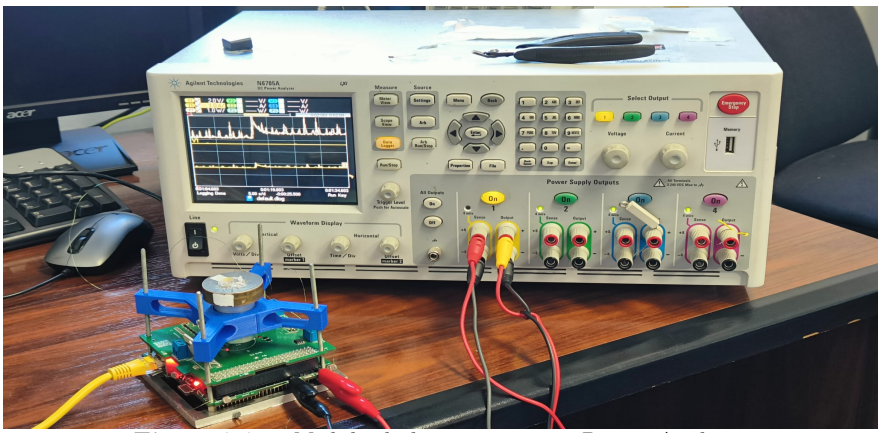


Figura 7.7 – *Medida de la potencia con Power Analyzer*

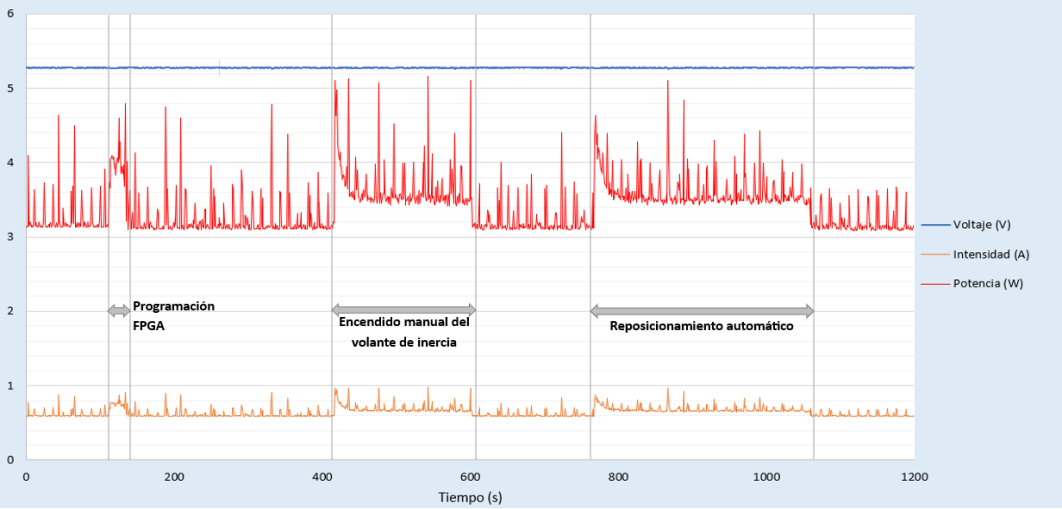


Figura 7.8 – *Potencia consumida por el CubeSat en operación*

La **potencia media** consumida en reposo, enviando telemetría y recibiendo telecomandos, es de 3,6 W. Esta potencia sube a 4,1 W cuando está activo el volante de inercia, siendo el instante de máximo consumo el arranque del motor DC debido al **pico de corriente** que se experimenta.

Capítulo 8

Conclusión y Futuras Líneas de Trabajo

Para concluir el presente Trabajo Fin de Grado se procederá a revisar los requerimientos definidos en el capítulo 3 para comprobar cuáles se han completado con éxito. Posteriormente se reflexionará sobre las líneas de mejora que se han descubierto a lo largo del desarrollo del proyecto con el fin de servir de inspiración tanto para un posible Trabajo Fin de Máster del autor, como para futuros trabajos o prácticas curriculares de otros estudiantes.

8.1 Cumplimiento de requerimientos

Ref.	Requerimiento	Cumplido
PM.Req.1	Reensamblaje del I2DOS y sustitución de posibles piezas deterioradas.	✓
PM.Req.2	Caracterización de los LEDs implementados en los simuladores solares.	✓
PM.Req.3	Búsqueda de un método de alimentación que permita la operación de forma segura.	✓
PM.Req.4	Diseño de un sistema de ventilación para la refrigeración de los simuladores solares.	✓

Tabla 8.1 – *Plataforma mecánica - Cumplimiento de requerimientos*

Ref.	Requerimiento	Cumplido
ET.Req.1	Diseño del panel de control de misión.	✓
ET.Req.2	Definición de paquetes de telemetría que permitan la transmisión de la información de los sensores del CubeSat.	✓
ET.Req.3	Definición de un listado de telecomandos que permitan alterar el comportamiento y actitud del CubeSat.	✓
ET.Req.4	Diseño de una interfaz de comunicación estable y fiable tanto cableada como inalámbrica que permita integrar uno o varios CubeSats.	✓

Tabla 8.2 – *Estación terrena - Cumplimiento de requerimientos*

Ref.	Requerimiento	Cumplido
CS.Req.1	Reposición o sustitución de sensores u otros elementos no implementados o defectuosos.	✓
CS.Req.2	Programación de un código que genere y envíe paquetes de telemetría en tiempo real a partir de los datos de los sensores.	✓
CS.Req.3	Programación de un código de recepción e interpretación de telecomandos.	✓
CS.Req.4	Establecimiento de las interfaces de conexión necesarias para la programación del OBC.	✓
CS.Req.5	Establecimiento de las interfaces de conexión necesarias para establecer una comunicación estable y fiable con la estación terrena.	✓
CS.Req.6	Restablecimiento de la comunicación inalámbrica con la estación terrena mediante la ESP8266.	✗
CS.Req.7	Programación de un algoritmo que identifique la dirección de la que proviene la luz solar.	✓
CS.Req.8	Programación de un código que permita controlar el volante de inercia para reposicionar el satélite tanto manualmente como de forma autónoma.	✓
CS.Req.9	Diseño de un soporte que resista el enorme par generado por el volante de inercia.	✓
CS.Req.10	El CubeSat debe consumir menos de 5 W en operación.	✓

Tabla 8.3 – *CubeSat - Requerimientos*

8.2 Futuras líneas de trabajo

Sin embargo, a pesar del trabajo realizado, el prototipo del CubeSat está lejos de estar completo. A continuación, se listan algunas de las líneas de trabajo más relevantes que emergen de la realización de este proyecto y que podrían contribuir al objetivo final de conseguir un producto terminado, capaz de desplegarse en órbita:

- Modelado de un **sistema de almacenamiento** de energía basado en **baterías** y **energía solar**.
- Diseño de una **armadura protectora** que resguarde la electrónica de las difíciles condiciones que se soportan en órbita.
- Implementación de Magnetopares para un control más preciso de actitud.
- Implementación de **nuevos sensores** enfocados a reunir datos de interés científico o práctico.
- Mejora de la plataforma mecánica para permitir simular nuevas variables como **perturbaciones en el campo magnético** o **cambios de temperatura**.
- Diseño de un **nuevo simulador inercial** que permita simular una **órbita en 3D**.
- Implementación de un control PID directamente la FPGA que permita una respuesta más rápida y precisa en el control de actitud.

En resumen, este Trabajo Fin de Grado marca el final de mi etapa de formación en el **Grado de Ingeniería Electrónica Industrial**, pero abre múltiples puertas para continuar adquiriendo nuevos conocimientos y habilidades en futuros estudios o proyectos en el ámbito universitario.

Bibliografía

[1] Durillo, J. C. M. Design of a multidisciplinary 1u cubesat simulation platform.

[2] ECSS. Estándar ecss-e-st-10c. https://ecss.nl/standard/ecss-e-st-10c-rev-1-system-engineering-general

[3] ESA. https://www.esa.int/.

[4] NASA. Insight lander. https://science.nasa.gov/mission/insight/.

[5] NASA. Nasa space science data coordinated archive. https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=2018-042B.

[6] RS. Led 14b10c datasheet. https://es.rs-online.com/web/.

[7] through Education in Science, C. C., and Research), A. Cubesat para educación. https://cesar.esa.int/index.php?Section=Cubesat_Control&ChangeLang=es#:~:text=El%20cubesat%20es%20el%20est%C3%A1ndar,un%20peso%20inferior%201.33%20kg.

[8] Wikipedia. Irradiancia solar. https://es.wikipedia.org/wiki/Irradiancia.

Anexos A

Documentación del Proyecto

Este **anexo** se centra en una de las **motivaciones principales** de este Trabajo Fin de Grado, que no se menciona durante el desarrollo del proyecto en los capítulos principales pero que se ha realizado paralelamente en sus diferentes etapas.

Esta motivación no es otra que la de **documentar** todos los procedimientos de ingeniería que se han ido aprendiendo y aplicando, en una serie ordenada de archivos de **Jupyter Notebook** con el fin de facilitar la aproximación a la materia de los CubeSats a **futuros estudiantes** que quieran orientar sus prácticas o trabajos fin de grado a este sector.

Los diferentes archivos están redactados y clasificados siguiendo un **orden de aprendizaje lineal** para que puedan utilizarse, junto con esta memoria, como guía introductoria a la materia. A continuación se muestra dicha clasificación, cada carpeta tiene asociado un archivo **Notebook** de extensión .ypynb y el material extra que se muestra.

```
Documentación CubeSat-I
├── 00_OBC_ARD_UNO
│   └── LED_BLink_COSMOS.ino
├── 01_XCTE_Protocol
├── 02_cFS_NASA
├── 03_OBC_ARD_MEGA
│   └── OBC_MEGA.ino
├── 04_Rasp_SSH
├── 04_Payload_Sensors
│   ├── SensorsTest.py
│   └── Librerias_Sensores
├── 06_ADCS_FPGA
│   ├── MotorTest.py
│   ├── FPGAprogrammer
│   └── FPGA_PWM
├── 07_Rasp_COSMOS
│   ├── Telemetry.py
│   └── Telecommands.py
└── 08_CubeSat-I_COSMOS
    ├── OBC.py
    └── CubeSat_1_COSMOS
```