UNIVERSIDAD DE CONCEPCIÓN

FACULTAD DE INGENIERÍA

DEPARTAMENTO INGENIERÍA MECÁNICA

INTEGRACIÓN DE UN CUBESAT BUS DISEÑADO COMO HERRAMIENTA DE APRENDIZAJE EN INGENIERIA DE SISTEMAS ESPACIALES
HAISE-Sat

POR

Bastián Philipe Villarroel Hernandez

Memoria de Título presentada a la Facultad de Ingeniería de la Universidad de Concepción para optar al título profesional de Ingeniero Civil Aeroespacial

Profesor Guía:
PhD (C) Alejandro Ignacio López Telgie

Marzo 2023
Concepción (Chile)

© 2023 Bastián Philipe Villarroel Hernandez

AGRADECIMIENTOS

Quiero dedicar este trabajo a mi familia, a mis padres Patricio y Maribel, quienes han estado ahí para mi desde el inicio, gracias a ellos, muchos de los conocimientos que poseo hoy en día han sido posibles, por su paciencia y entrega durante todo mi proceso formativo, desde que ingrese a la etapa escolar, hasta hoy que termino una etapa importante de mi vida, y que se proyecta a mi futuro profesional donde seguirán tomando un rol importante en lo que será mi vida. Quiero agradecer a mis amigos aeroespaciales y electrónicos en la realización de este trabajo, recibí retroalimentación de ellos durante todo el periodo de realización de este. En especial quiero agradecer a mis amigos Felipe, Seba y Pedro por estar constantemente atendiendo mis inquietudes al momento de redactar este trabajo, sin ellos esto no hubiera sido posible.

Tengo un gran agradecimiento a mis profesores de mi formación técnica, en especial a un profesor muy querido por mi generación, Jorge Roa, Q.E.P.D. y a Cristian Carrasco, hay mucho de mi manera de ver el mundo que puedo atribuir de forma positiva a la interacción que tuve con ellos esos 2 años como Técnico Electrónico.

Un agradecimiento especial a un amigo del liceo, Luciano, con el cual retomé contacto hace pocos años, y quien estuvo ahí para comentar y discutir la mayor parte de la programación de mi proyecto, realmente fue de gran ayuda para cada una de las iteraciones que realicé en el Software de mi trabajo.

Quiero destacar la importancia de mi experiencia como practicante en el Grupo de Operaciones Espaciales de la Fuerza Aérea de Chile, mucho de lo que aprendí en esta instancia, pude aplicarla a este trabajo para obtener un producto que tuviera un peso en el objetivo que debía cumplir, mis agradecimientos son para el personal del GOE, quienes con paciencia y dedicación, se encargaron de entregarme conocimientos importantes en el ámbito de sistemas satelitales, esta experiencia aportó de gran manera al diseño del Sistema de este proyecto.

Para finalizar, quiero agradecer a todos los profesores que atendieron mis dudas en estos 6 años de carrera, en especial a Don José, puedo decir que me ayudó a pulir varios aspectos de mi persona y al profesor Alejandro quién hizo posible la realización de este proyecto, estando presente desde el inicio con el Proyecto de Ingeniería, gracias por depositar su confianza en mi para desarrollar este proyecto.

No podría irme sin agradecer a mi perrita Coka, quien en la época de pandemia estuvo a mi lado cuando más lo necesite, en esas instancias donde la interacción con el mundo exterior se limitaba a unas pantallas, fue ella quien me permitió distraerme del cansancio mental y seguir adelante con mis estudios, jamás te olvidaré.

RESUMEN

Bajo el contexto de la creciente industria espacial y el impacto que han tenido los *Small Satellites* en las formas de acceder a esta industria, aparecen los CubeSats, satélites pequeños cuyo formato queda bien definido por un estándar de tamaño, forma y masa, que, sumados a los desarrollos de la industria de lanzadores espaciales, resultan en la reducción de los costos para acceder al espacio lo cual ha permitido que la academia pueda llevar a cabo actividades en órbita terrestre y más allá. Estos satélites contemplan varias características, que pueden ser aterrizadas a herramientas para entrenamiento en sistemas espaciales sin la necesidad de llevarlos a órbita, siendo un punto de partida clave cuando se busca desarrollar masa crítica de profesionales en el área. Con esto en mente, surge la idea de desarrollar una herramienta de aprendizaje que simule las cualidades de un satélite tipo CubeSat con un costo menor a los satélites diseñados para ir a órbita, pero que refleje bien la arquitectura y funcionalidades de un CubeSat real, esto con el fin de que pueda ser implementado en cursos académicos o en cursos de capacitación profesional.

Este proyecto se dividió en dos partes, la primera contemplada en el Proyecto de Ingeniería Aeroespacial, donde el satélite de entrenamiento es diseñado, definiéndose cuales son las capacidades que deberá tener, mediante la identificación de una arquitectura física y funcional atacada desde el punto de vista del *Systems Engineering*, el resultado de esta etapa consistió en una propuesta de componentes para la integración del satélite basado en componentes *Commercial off-the-shelf,* COTS, los cuales son una cualidad importante de los CubeSat reales, lo que ha permitido reducir costos y facilitar el acceso al espacio de instituciones académicas y centros de investigación.

La segunda parte de este proyecto corresponde a esta memoria de título, la cual consiste en la fabricación de esta Herramienta de Aprendizaje en Ingeniería de Sistemas Espaciales, lo que lleva a nombrar a este satélite de entrenamiento, HAISE-Sat, por sus siglas en español. Este trabajo contempla una evaluación del diseño preliminar, realizando una nueva revisión de esta y efectuando cambios que permitan la integración del satélite, tomando en cuenta los requisitos que tendrá para que sea de utilidad como herramienta de apoyo para el aprendizaje del estudiante. Esta revisión es realizada en cada subsistema del HAISE-Sat, donde los principales cambios son efectuados en el Subsistema de Potencia dados los inconvenientes encontrados al verificar la viabilidad del diseño de forma experimental, adicionalmente se genera un diseño del Subsistema Estructural el cual no estaba presente en el diseño preliminar.

Posteriormente a la integración y pruebas del satélite de entrenamiento se identifican experiencias de aprendizajes presentes en la literatura y se propone una guía de actividades tomando en cuenta las capacidades de esta primera versión. Por otro lado, este satélite de entrenamiento es la suma de Hardware y Software, por lo que se dedica parte de este trabajo a describir la lógica utilizada en la programación, destacando el hecho de que está todo programado en el lenguaje Python, volviendo a este satélite una herramienta versátil según las necesidades del docente, entendiendo esto, se ve un gran potencial de actualización y mejora en el Software para simular de mejor manera un CubeSat Real.

Tabla de Contenidos

Tabla de Contenidos i
Lista de Tablas iii
Lista de Figuras iv
Glosario vii
1 CAPÍTULO 1: Introducción 1
1.1 Planteamiento del problema 7
1.2 Propuesta de solución a la problemática 7
1.3 Objetivos generales y específicos 10
1.4 Hipótesis 10
1.5 Metodología de Trabajo 12
1.6 Metodología de trabajo para el diseño de la experiencia de aprendizaje. 15
2 CAPITULO 2: Marco teórico 16
2.1 Arquitectura de un Small Satellite. 16
2.2 Lógica de integración de los subsistemas 20
3 CAPITULO 3: Diseño y prototipado del HAISE-Sat 22
3.1 Software utilizado. 22
3.2 Subsistema Estructural 23
3.3 Asignación de pines al conector eléctrico. 31
3.4 Subsistema de comandos y manejo de datos (C&DH). 34
3.5 Payload 39
3.6 Subsistema de comunicaciones 40
3.7 Subsistema de Potencia 40
3.8 Prototipado del HAISE-Sat en Protoboard. 48
3.9 Diseño del *Backplane*, placa de conexiones. 49
3.10 Integración del HAISE-Sat. 49
4 CAPITULO 4: Propuesta de experiencia de aprendizaje 51
4.1 Estructura de la experiencia 51
4.2 Primera etapa: Entrega de conocimientos 52
4.3 Segunda etapa: Experiencia *Hands-on* 52
5 CAPÍTULO 5: Software del HAISE-Sat 55

5.1 Software de vuelo a bordo del HAISE-Sat 55
5.2 Software para implementar la estación de control 56
6 CONCLUSIONES 58
Referencias 61
7 ANEXO 66

Lista de Tablas

Tabla 1-1 Nomenclatura para *Small Satellites* del programa *Small Spacecraft Technology* (SST) de la NASA (NASA, 2020) 3
Tabla 1-2 Taxonomía para satélites según masa y tamaño para *Small Satellites*, propuesta de (Botelho A. S. & Xavier, 2019) 6
Tabla 3-1. Pines con señal asociadas comunes para HAISE-Sat basado en Proyecto de ingeniería. 32
Tabla 3-2. Asignación de pines 33
Tabla 3-3 Opciones regulador de voltaje 5 V. 42
Tabla 3-4. Resultados de mediciones en las pruebas del subsistema de potencia. 48
Tabla 3-5. Verificación de Requisitos y Restricciones. 50
Tabla 4-1. Guía de actividades para el HAISE-Sat, propuesta inicial. 53
Tabla 7-1 Comportamiento de ADA390. 66
Tabla 7-2. Distinción de identificador para sensor INA219 en protocolo I2C 70

Lista de Figuras

Figura 1-1. Jerarquía de un sistema según ECSS (ECSS Secretariat, 2012). 1
Figura 1-2. Segmentos del sistema espacial definido para la misión del satélite RaioSat desarrollada por el *Instituto Nacional de Pesquisas Espaciais*, INPE (Filho et al., 2020)..... 2
Figura 1-3. a) Acoplamiento de 3 ESPA de la empresa Moog. b) Satélites de la empresa ORBCOMM montados en los ESPA. Imágenes recuperadas de (ORBCOMM, 2016). 3
Figura 1-4. ISIPOD, *Picosatellite Orbit Deployer* de ISISPACE en distintos formatos de tamaño (CubeSatShop, 2022b). 4
Figura 1-5. a) POD siendo montados en un cohete Electron de Rocket Lab, corresponde a la misión ELaNA-19 de la NASA (Rocket Lab, 2022). b) UWE-4: CubeSat 1U desarrollado en la Universidad de Wurzburg (Kramer & Schilling, 2021) 4
Figura 1-6. Resultados de estudio de Bryce Tech, distribución de satélites por masa lanzados en el periodo 2012-2021 (Bryce Space and Technology, 2022)..... 5
Figura 1-7. Resultados de estudio de Bryce Tech, distribución de satélites por aporte de masa al total de satélites lanzados (Bryce Space and Technology, 2022)..... 6
Figura 1-8. Ejemplo de estructuras de CubeSats en diferentes factores de forma (Poghosyan & Golkar, 2017)..... 9
Figura 1-9. *Product Breakdown Structure* (PBS) del *RaioSat Satellite* , indicando la procedencia de cada componente (Filho et al., 2020). 9
Figura 1-10. Descomposición funcional (FBS) del CubeSat Bus a construir, elaboración propia. ... 10
Figura 1-11. Subsistemas del HAISE-Sat obtenidos del diseño realizado en el PIA..... 11
Figura 1-12. Breadboard utilizada para realizar conexiones en una Raspberry Pi Zero, imagen recuperada de (Ashish MIshra, 2021). 15
Figura 2-1. Descomposición funcional de un satélite (Kiselyov, 2020) 16
Figura 2-2 Descomposición en productos de un satélite (Kiselyov, 2020)..... 17
Figura 2-3. Lógica de integración de satélites del programa BIRDS..... 20
Figura 3-1. *Product Breakdown Structure* del HAISE-Sat en el diseño preliminar 22
Figura 3-2. Modelo tridimensional de CubeSat 1U de ISISPACE (ISISPACE, 2022), archivo .step visualizado con Inventor de Autodesk. 24
Figura 3-3. Modelo tridimensional original de ArduSat (MAKERFAM, 2012) , archivo .stl visualizado con Inventor de Autodesk. 24
Figura 3-4. ArduSat impreso en la facultad, con una representación de sus subsistemas. 25
Figura 3-5. Bases de la estructura del ArduSat 25
Figura 3-6. Fijaciones laterales de la estructura. 25
Figura 3-7. Comparación visual de modelo modificado y modelo original..... 26
Figura 3-8. Primer diseño fabricado de la estructura. 26
Figura 3-9. Unión por interferencia entre fijaciones laterales y bases de la estructura, dimensiones en mm..... 27
Figura 3-10. Modificación a fijación lateral..... 28
Figura 3-11. Integración de la segunda iteración del diseño estructural. 28

Figura 3-12. Acople para integrar un panel solar a la estructura. 29
Figura 3-13. Integración del panel solar a la estructura. 29
Figura 3-14. Modelo 3D del subsistema de potencia del ArduSat 30
Figura 3-15. Proyección del contorno y agujeros del subsistema de potencia del ArduSat. 30
Figura 3-16. Geometría exportada a EasyEDA para diseño de placas de circuito. 31
Figura 3-17. Integración de placas de subsistemas en satélites BIRDS (Kumar et al., 2018). 31
Figura 3-18. Asignación de señales en los pines de la Raspberry Pi Zero (Raspberry Pi Foundation, 2018b) 32
Figura 3-19. Distribución de pines para el conector del HAISE-Sat 33
Figura 3-20. Aspecto de la Raspberry Pi Zero (Raspberry Pi Foundation, 2018a). 34
Figura 3-21. Primera iteración diagrama eléctrico para el subsistema de comandos y manejo de datos C&DH. 35
Figura 3-22. Placa de circuitos del C&DH, vista desde la cara superior. a) Capa superior, b) Capa inferior. 36
Figura 3-23. Placa de circuitos del C&DH fabricada. 36
Figura 3-24. Prototipado del montaje de los componentes del subsistema C&DH. 37
Figura 3-25. Dimensionamiento del soporte para los componentes del subsistema C&DH usando un modelo 3D de una Raspberry Pi Zero. 37
Figura 3-26. Generación de Sketch para diseño de apoyo de componentes del subsistema C&DH. 38
Figura 3-27. Modelo 3D del soporte para los componentes del subsistema C&DH. 38
Figura 3-28. Integración del subsistema C&DH. 39
Figura 3-29. Subsistema de potencia del diseño preliminar. 40
Figura 3-30. a) Módulo de carga solar ADA390, b) Regulador de voltaje XL6009 41
Figura 3-31. Regulador de 5V y 3.3V (Adafruit, 2022a) 43
Figura 3-32. Sensor de corriente y voltaje INA219. 43
Figura 3-33. Diseño eléctrico del subsistema de potencia. 44
Figura 3-34. Primera versión de la placa de circuitos del subsistema de potencia fabricada. 45
Figura 3-35. Batería Li-Ion de referencia (Papis et al., 2020). 45
Figura 3-36. Porta baterías 18650, recuperado de (Thingiverse, 2022). 46
Figura 3-37. Porta batería compatible con la estructura. 47
Figura 3-38. Integración del subsistema de potencia 48
Figura 3-39. Prototipado del diseño eléctrico en Protoboard. 49
Figura 3-40. HAISE-Sat integrado. 50
Figura 4-1. Flujo de trabajo utilizado en el programa HEPTA-Sat (Yamazaki & Zengo, 2018). 51
Figura 5-1. Estructuración del programa que ejecuta HAISE-Sat. 56
Figura 5-2. Estructura del programa que ejecuta la estación de control. 57
Figura 7-1. Configuración para pruebas del XL6009 68
Figura 7-2. Mediciones para distintos voltajes de entrada en el XL6009 68
Figura 7-3. Verificación del XL-6009 con tensión inferior al indicado por el fabricante. 69
Figura 7-4. Diagrama de conexión del sensor INA219 69
Figura 7-5. Diseño y Fabricación de la placa *Backplane* 70

Figura 7-6. Componentes del Kit HAISE-Sat 71
Figura 7-7. Secuencia de integración del Hardware del HAISE-Sat 75
Figura 7-8. Estado de la batería en el Modo 1. 93
Figura 7-9. Estado de la batería en el Modo 2. 93
Figura 7-10. Dimensiones del CubeSat Bus fabricado. 94
Figura 7-11. Masa del HAISE-Sat medida con una balanza de cocina. 95

Glosario

AOCS	:	*Attitude and Orbit control subsystem*
C&DH	:	*Command & Data Handling subsystem*: Manejo de comandos y datos
Cal Poly	:	Universidad Politécnica Estatal de California
CDS	:	CubeSat *Design Specification*
ConOps	:	Concepto de operaciones
COTS	:	*Commercial off-the-shelf*
CSID	:	CubeSat *Subsystem Interface Definition*
DC	:	Corriente continua
EELV	:	*Evolved Expendable Launch Vehicle*
ELaNA	:	*Educational Launch of NanoSatellites*
EPS	:	*Electric Power subsystem*
ESPA	:	*EELV Secondary Payload Adapter*
FBS	:	*Function Breakdown Structure*
GND	:	Tierra o Ground, generalmente es terminal negativo de fuente de poder.
GPS	:	*Global Positioning System* - Sistema de Posicionamiento Global
GS	:	*Ground Segment*
HAISE	:	Herramienta de aprendizaje en Ingeniería de Sistemas Espaciales
INPE	:	*Instituto Nacional de Pesquisas Espaciais*
Kyutech	:	Instituto de tecnología de Kyushu
NASA	:	*National Aeronautics and Space Administration*
OBC	:	*On Board Computer*
PBS	:	*Product Breakdown Structure*
PCB	:	Printed Circuit Board, Placa de circuito impreso
PIA	:	Proyecto de Ingeniería Aeroespacial
POD	:	*Picosatellite Orbital Deployer*
PPOD	:	*Poly-Picosatellite Orbital Deployer*
SBC	:	*Single Board Computer*
SeBOK	:	*Systems Engineering Body of Knowledge*
SST	:	*Small Spacecraft Technology*
STEM	:	*Science, Technology, Engineering and Mathematics*
TRL	:	*Technology Readiness Level*
TT&C	:	*Telemetry, Tracking & Command*
UNISEC	:	*University Space Engineering Consortium*
VCC	:	Fuente de poder, terminal positivo
IMU	:	*Inertial Measurement Unit*

CAPÍTULO 1: Introducción

Hoy en día nos movemos en un mundo impulsado por el desarrollo tecnológico que ha logrado el ser humano con el pasar de los años, un aspecto importante de ello ha sido la miniaturización de la electrónica en los sistemas que usamos a diario, se ha logrado reducir el espacio requerido por la electrónica y expandido las capacidades de esta, se ha logrado reducir el volumen y la masa de muchos sistemas, a la vez que se ha aumentado o mantenido sus capacidades al incorporar las nuevas tecnologías en su diseño y fabricación (Iwai, 2021; Madry et al., 2018; Poghosyan & Golkar, 2017). Un sector que se caracteriza por aprovechar los beneficios de las innovaciones tecnológicas es la industria espacial, se observa en varios ámbitos como ciencias de materiales, procesos de fabricación, comunicaciones, energía entre otros (Bockel, 2018), y para efectos de este trabajo, el impacto que han tenido todas estas innovaciones en los sistemas espaciales, desde mejoras en el almacenaje de energía, obtención de esta, hasta la optimización de la potencia necesaria por las computadoras que operan en órbita entre otras, incluyendo todas las mejoras de infraestructura de estos sistemas para su operación desde tierra. Esto ha permitido que estos sistemas vayan mejorando su desempeño, expandiendo sus capacidades o abaratando sus costos, todo este contexto ha dado lugar a que se manifiesten una clase particular de satélites en los sistemas espaciales conocidos como *Small Satellites* cuya filosofía de diseño se expondrá más adelante.

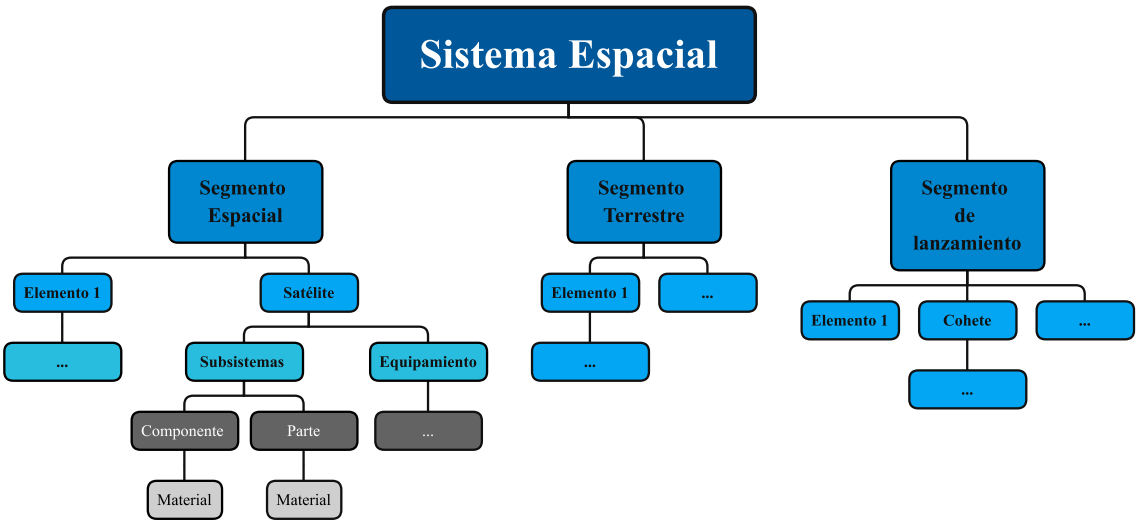


Figura 1-1. Jerarquía de un sistema según ECSS (ECSS Secretariat, 2012).

Para poner en contexto, un Sistema espacial se compone de segmentos, siendo los más generales el Segmento Terrestre, Segmento Espacial y Segmento de Lanzamiento, estos segmentos se componen de elementos, estos elementos se componen de subsistemas o equipamiento, a su vez estos están

constituidos por Componentes o Partes [1], llegando a un nivel inferior marcado por los Materiales en lo cual está fabricado cada cosa que constituyen al Sistema Espacial (ECSS Secretariat, 2012). Una vista general de esta jerarquía se observa en la Figura 1-1.

Un ejemplo de la descomposición del sistema a nivel de segmentos se observa en el RaioSat, un CubeSat 3U (nomenclatura que se indicará más adelante) en la Figura 1-2.

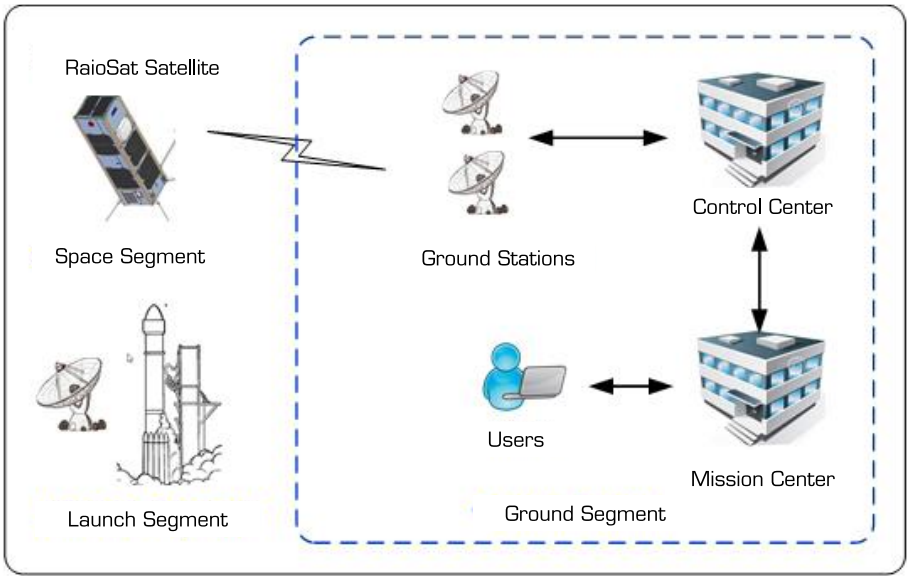


Figura 1-2. Segmentos del sistema espacial definido para la misión del satélite RaioSat desarrollada por el *Instituto Nacional de Pesquisas Espaciais,* INPE (Filho et al., 2020)**.**

Un ejemplo de Elementos de un Segmento Espacial son los Satélites (*RaioSat Satellite* en la Figura 1-2), estos desempeñan la función de cumplir con una misión determinada, estas misiones pueden ser proveer comunicaciones, televisión satelital, internet, observación terrestre o percepción remota, cada una de ellas tendrá que satisfacer distintas necesidades mediante una combinación de hardware, software, características de la órbita, instalaciones en tierra, personas, servicios de apoyo, etc. En particular, dentro de los satélites existen distintas nomenclaturas para definir qué tipo es cada satélite, el enfoque de este trabajo está en los *Small Satellites*, cuya definición varía según a que institución se le consulte, una de ellas es la provista por la NASA indicando según la masa el tipo de satélite. En particular para una caracterización de los *Small Satellites*, se observa la Tabla 1-1.

[1] Componente es mayormente utilizado para referirse a Electrónica mientras que Parte es utilizado para algo mecánico, estructural, etc.

Tabla 1-1 Nomenclatura para *Small Satellites* del programa *Small Spacecraft Technology* (SST) de la NASA (NASA, 2020) **.**

Nomenclatura de *Small Satellites*	Rango de Masa [kg]	
	Inferior	**Superior**
Minisatellite	100	180
Microsatellite	10	100
Nanosatellite	1	10
Picosatellite	0.01	1
Femtosatellite	0.001	0.01

Para reducir los costos de lanzamiento y optimizar otros aspectos (Goodwin & Wegner, 2001), con los años se ha ido generando una estandarización de las interfaces entre satélite y vehículo lanzador, parte de esta lógica de estandarización se puede observar en los mecanismos que se encargan de desplegar a los satélites en órbita, en la Figura 1-3 se puede ver anillos que se montan en la última etapa del cohete, permitiendo instalar los sistemas de despliegue de *Small Satellites*.

a) **b)**

Figura 1-3. a) Acoplamiento de 3 ESPA de la empresa Moog. b) Satélites de la empresa ORBCOMM montados en los ESPA. Imágenes recuperadas de (ORBCOMM, 2016).

Por parte de la caracterización de NASA en la Tabla 1-1, en los últimos años esta ha quedado insuficiente dado el surgimiento de un tipo de *Small Satellite* denominado CubeSat, estos satélites pequeños están sujetos a un estándar que pone restricciones a su forma, volumen y masa, con tal de facilitar el proceso de pruebas y montaje en los vehículos de lanzamiento entre otros aspectos.

Sin embargo, para el caso de los CubeSats, es el propio sistema de despliegue el que determina el volumen y forma de estos satélites, se puede ver en la Figura 1-4 un ejemplo de *Pico Satellite Deployer* (POD), estos dispositivos se acoplan como carga útil secundaria de misiones con satélites más grandes a los vehículos de lanzamiento, lo que a su vez ha permitido reducir los costos de lanzamiento y popularizar a los CubeSat como oportunidad para probar tecnología en órbita, realizar investigación científica, o que estudiantes de carreras de ingeniería o afines lleven a cabo misiones en

el espacio con tal de desarrollar habilidades y experiencia en ingeniería de sistemas espaciales o experimentar en el espacio (Woellert et al., 2011).

a) **b)**

Figura 1-4. ISIPOD, *Picosatellite Orbit Deployer* de ISISPACE en distintos formatos de tamaño (CubeSatShop, 2022b)**.**

En la Figura 1-5 se tiene el montaje de 13 CubeSats desarrollados bajo la misión ELaNa-19 (*Educational Launch of NanoSatellites 19*), consistió en el lanzamiento de 10 CubeSat para investigación desarrollado por Universidades y 3 CubeSat desarrollados en Centros de la NASA con participación de una escuela secundaria. La importancia es que participaron cerca de 250 estudiantes en las etapas de diseño, desarrollo y construcción de estos satélites, muchos de ellos desarrollando actividades de investigación científica o como parte de sus programas académicos. La particularidad de estas misiones (misiones ELaNA-##) es que son llevadas a cabo bajo un programa (*Venture Class Launch Services*, programa de la NASA) enfocado en fomentar el uso y desarrollo de vehículos de lanzamiento dedicados a cargas útiles pequeñas como lo son los *Small Satellites* (NASA, 2018) y que ha permitido acercar la ingeniería satelital a los estudiantes involucrados de una forma mucho más aplicada y relevante para su proceso de formación como profesionales del área.

a) **b)**

Figura 1-5. a) POD siendo montados en un cohete Electron de Rocket Lab, corresponde a la misión ELaNA-19 de la NASA (Rocket Lab, 2022)**. b) UWE-4: CubeSat 1U desarrollado en la Universidad de Wurzburg (Kramer & Schilling, 2021)**

Observando el análisis realizado por la empresa Bryce Tech (Bryce Space and Technology, 2022) en la Figura 1-6 , el crecimiento del sector satelital ha tenido como protagonistas a los *Small Satellites*, fuertemente influenciado por las actividades de gigantes como SpaceX con Starlink, o OneWeb, ambos orientados a proveer internet satelital, estos lanzan una gran cantidad de satélites anualmente y demuestran las capacidades reales de sistemas basados en satélites pequeños que forman una red con cierto grado de redundancia. En el caso particular de estas dos empresas, los satélites de Starlink tienen una masa que ronda los 260 kg (Gunter's Space Page, 2022c) mientras que OneWeb posee satélites que aproximan los 147 kg (Gunter's Space Page, 2022a), dicho esto, solo OneWeb entraría en la categoría de *Small Satellite* de NASA. Luego si se comparan estas masas con satélites más pesados del orden de magnitud de las toneladas como el *Radarsat-2* con 2300 kg (Gunter's Space Page, 2022b) que opera en órbita baja (cuya misión es de observación terrestre a 787 km sobre el nivel del mar), o uno de los satélites GPS (*Global Positioning System*) del sistema *Navstar-3* con una masa de 4400 kg (a 20200 km de altura sobre el nivel del mar), los *Small* Satellites son más pequeños como su nombre indica, además, estos satélites grandes nombrados cuya masa llega a las toneladas, muchas veces operan de forma solitaria y una falla puede ser crítica para el desempeño de su misión lo cual permite identificar algunas ventajas al diseñar sistemas basados en satélites pequeños, una de estas ventajas es que los *Small Satellites*, al requerir de menos tiempos de desarrollo, permiten actualizar y mantener la arquitectura del sistema con mayor frecuencia que con los satélites más pesados (Madry et al., 2018).

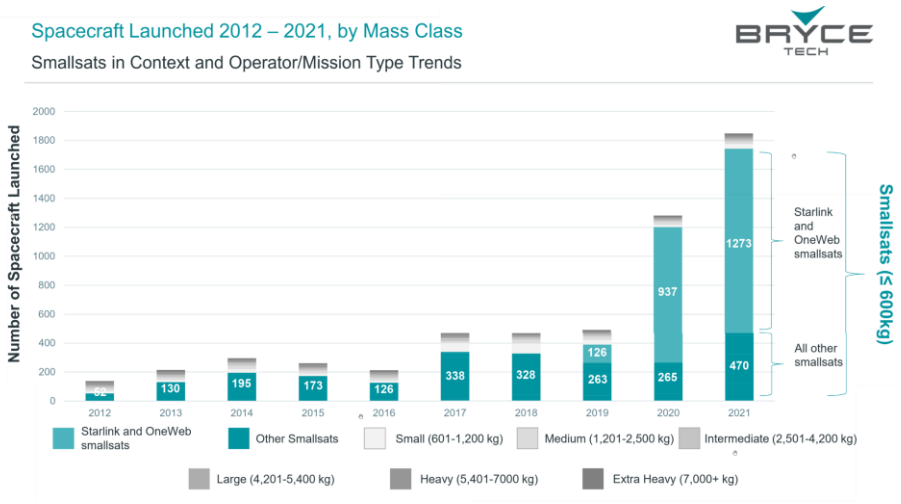


Figura 1-6. Resultados de estudio de Bryce Tech, distribución de satélites por masa lanzados en el periodo 2012-2021 (Bryce Space and Technology, 2022)**.**

En la Figura 1-7 se puede observar el total de satélites lanzados hasta la fecha indicada, se observa que los *Small Satellites* comienzan a equiparar el "reducido" numero de satélites más grandes en términos de masa

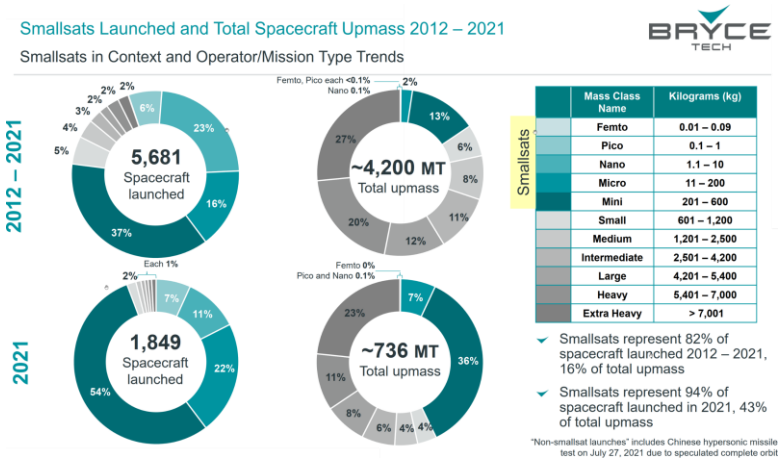


Figura 1-7. Resultados de estudio de Bryce Tech, distribución de satélites por aporte de masa al total de satélites lanzados (Bryce Space and Technology, 2022).

Tabla 1-2 Taxonomía para satélites según masa y tamaño para *Small Satellites*, propuesta de (Botelho A. S. & Xavier, 2019)

#	Clase		Subclase			*Tamaño*
	Nombre	**Masa (kg)**	**Tipo**	**Masa**		
3	Mini	[100 - 1000[	*Heavy*	[500 - 1,000[		*Small*
			Intermediary	[180 - 500[		
			Light	[100 - 180[		
2	Micro	[10 - 100[	*Heavy*	[60 - 100[		
			Intermediary	[25 - 60[		
			Light	12U	[10 - 25[	
1	Nano	[1 - 10[		6U	[8 - 10[	
					[6 – 7.99[	
			Cubesat	3U	[3 - 3.99[	
				2U	[2 - 2.66[	
				1U	[1 - 1.33[	
			Otra forma	[1 - 10[		
0	Pico	[0.1 - 1[	-	-		*Very Small*
-1	Femto	[0.01 – 0.1[	-	-		*Ultra-Small*
-2	Gram	[0.001 – 0.01[	-	-		*Ultra-very small*

1.1 Planteamiento del problema

Una de las principales barreras que se observan en el aprendizaje de tópicos relacionados al desarrollo y operación de misiones espaciales es el costo para llevar a cabo dichas misiones tradicionales como lo pueden ser sistemas espaciales de navegación global, comunicaciones u observación de la tierra entre otros. Un caso concreto es el SSOT con el satélite FASat Charlie, cuya misión es de observación terrestre, fue desarrollado por EADS Astrium y se entrenó a personal Chileno para la operación de la estación terrestre instalada en la base de la Fuerza Aérea de Chile El Bosque, el costo de este proyecto según fuentes se sitúa en el orden de magnitud de 70 millones de dólares (Torres, 2017), por otra parte, este satélite tuvo una masa de lanzamiento de aproximadamente 130 kg por lo que entra en la categoría de *Small Satellite*.

El costo de este proyecto resulta elevado para situarlo en ambientes académicos y que sean reproducibles con frecuencia para que personal pueda obtener experiencia en este ámbito, por lo que se necesita buscar alternativas a un menor costo, que permita que el estudiante interesado en Sistemas Espaciales tenga la chance de adquirir una base de experiencia y habilidades mientras realiza sus estudios, estas cualidades deben ser afines a los requerimientos solicitados para un profesional de la industria en la que están inmersos estos sistemas. Estas alternativas deben ser además de relevantes para el perfil como profesional, ser financiables por la institución en la que se desempeña el alumno, con tal de que se vuelva un recurso asequible para diversas casas de estudio, y una vez con esta base de conocimientos y habilidades, pueda desempeñarse en proyectos de mayor envergadura como lo fue el FASat Charlie, obviando el hecho de que este en particular fue integrado completamente en el extranjero. Llevando esta situación a un escenario nacional, el Sistema Nacional Satelital, SNSat (MinCiencia, 2022), contempla la integración de satélites en Chile y es ahí, donde una herramienta a bajo coste de implementar para formar profesionales aptos para trabajar en el SNSat es imprescindible.

1.2 Propuesta de solución a la problemática

Una opción a experiencias de bajo costo (relativo a misiones convencionales) surgió en 1999 cuando se propuso un estándar de satélites que permitía incorporar su desarrollo y operación en casas de estudio como Universidades, esto se lograba con el tamaño, la masa y la lógica de integración, ya que al estar bajo un estándar, los costos de producción de cada pieza requerida para la integración de estos podía verse reducida, además por estas ventajas expuestas, los tiempos de desarrollo permitían incorporar el ciclo de vida completo de uno de estos satélites en el tiempo que dura una carrera universitaria orientada a *Science, Technology, Engineering and Mathematics* (STEM), estos satélites corresponden a los CubeSat, satélites pequeños que siguen un estándar el cual delimita el factor forma, masa, tamaño e interfaces entre otros aspectos. Quien iniciaría este nuevo concepto de un satélite estandarizado con las características mencionadas fue la Universidad Politécnica Estatal de California, CalPoly, esto lo hizo mediante la propuesta de un sistema de despliegue que permitiría acoplar estos pequeños satélites como carga útil secundaria a vehículos de lanzamiento, así al estandarizar una

interfaz entre el satélite y el cohete, facilitaría el desarrollo de estos CubeSats para cumplir plenamente con las exigencias de las empresas dedicadas a enviar satélites al espacio y minimizar los riesgos y probabilidades de falla, que podrían poner en riesgo la misión del vehículo de lanzamiento. Este estándar se resume en el CubeSat *Design Specification* (CDS) publicado por Cal Poly. La primera versión publicada indica que un CubeSat 1U corresponde a un cubo de 10 x 10 x 10 cm^3 con una masa no mayor a 1,33 kg y que además obedece algunas condiciones de diseño impuestas para lograr la integración con el *Picosallite Orbital Deployer* (POD), en la revisión actual del documento, la revisión 14 del CDS (Cal Poly, 2020), se detallan ciertas características que deben tener los CubeSat para facilitar la integración en los vehículos de lanzamiento disponibles, además de señalar la escalabilidad que ha tenido el factor forma 1U para obtener CubeSat 3U, 6U, 12U, etc., permitiendo obtener un estándar de satélite más flexible ante las necesidades impuestas por una misión determinada.

Para la formación de estudiantes enfocados en los sistemas espaciales, Aaron Boaf et al (Aboaf et al., 2020) señala que, hacer que los alumnos interactúen con *hardware* como lo son los CubeSat al llevar a cabo el desarrollo de misiones en sus programas de estudio, tiene efectos bastante positivos en la experiencia y habilidades que obtienen para su perfil profesional, otro punto que menciona es que la formación mediante la comprensión de cada subsistema al descomponer un satélite y luego analizar la interacción de estos al realizar la integración y pruebas, resulta un buen método en términos de resultados de aprendizaje logrados al final de las experiencias.

Otro autor, Kenjiro et al (Lay et al., 2022), indica que los CubeSat son una buena herramienta para proveer a alumnos con experiencias *hands-on* en diseño, construcción y pruebas en el área satelital, al poder realizarse a un bajo costo (tomando como referencia el costo asociado al ciclo de vida de satélites tradicionales que pueden llegar a superar el millón de euros con facilidad (European Space Agency, 2022)), esto gracias a la aparición de los componentes *Commercial off-the-shelf* (COTS) que facilitan la integración de estos satélites tipo CubeSat al contar con interfaces estandarizadas, existiendo ya componentes con experiencia de vuelo relevante a un costo reducido, lo cual reduce los tiempos de pruebas y desarrollo previos al lanzamiento de estos a orbita, reduciendo así las barreras para poder llevar a cabo actividades en el espacio, además de la reducción de los costos de lanzamiento para satélites que van integrados como carga útil secundaria.

Así, con este trabajo se busca dar continuidad a lo propuesto en el Proyecto de Ingeniería Aeroespacial, PIA (Villarroel, 2022), lo cual consiste en desarrollar una herramienta que permita llevar a cabo una experiencia de aprendizaje para los alumnos que buscan desempeñarse en sistemas espaciales, interactuando y manipulando hardware con tal de formar una base de conocimientos sobre los sistemas espaciales y los procesos que se utilizan en esta industria. Con esta primera versión de la herramienta de aprendizaje, este satélite de entrenamiento llamado HAISE-Sat, se busca introducir conceptos de ingeniería de sistemas espaciales tales como los distintos niveles que hay en la jerarquía de estos sistemas, específicamente a nivel de elemento de un sistema espacial. Un ejemplo de la descomposición física de un satélite tipo CubeSat se puede observar en la Figura 1-9, destacar la presencia de los componentes COTS, que es una de las filosofías del diseño de este proyecto.

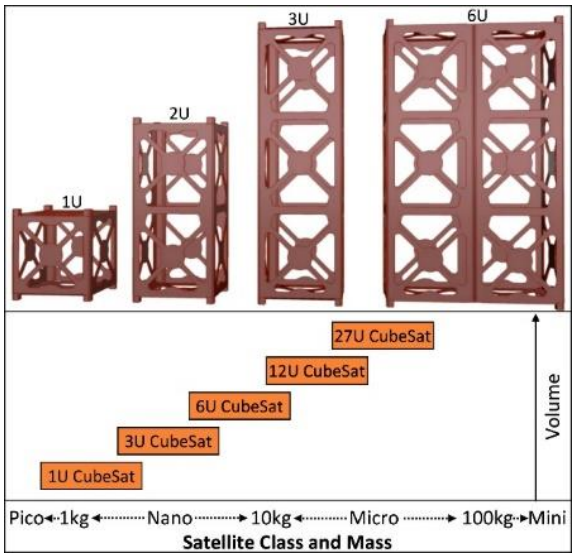


Figura 1-8. Ejemplo de estructuras de CubeSats en diferentes factores de forma (Poghosyan & Golkar, 2017).

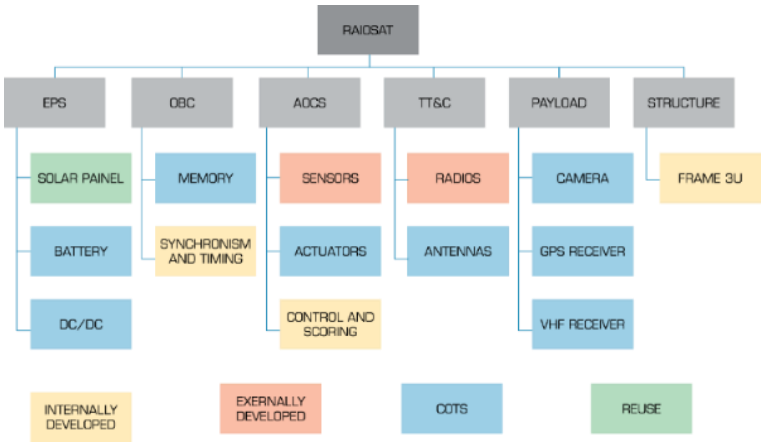


Figura 1-9. *Product Breakdown Structure* (PBS) del *RaioSat Satellite* , indicando la procedencia de cada componente (Filho et al., 2020).

Finalmente, basado en el PIA (Villarroel, 2022), se puede resumir en la Figura 1-10 una descomposición funcional o *Function Breakdown Structure* de la solución al problema, que consistirá en un satélite de entrenamiento, esta descomposición es generada a partir de las restricciones y requisitos que se expusieron en el PIA, esta figura condensa la composición del CubeSat Bus.

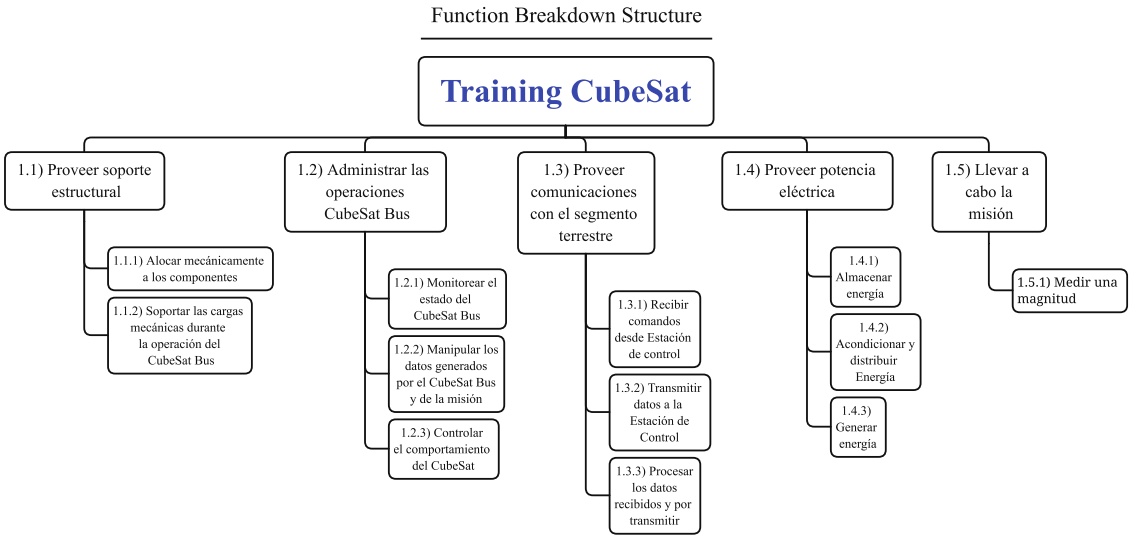


Figura 1-10. Descomposición funcional (FBS) del CubeSat Bus a construir, elaboración propia.

1.3 Objetivos generales y específicos

Objetivo general:

Desarrollo de la integración, verificación y validación de un sistema de apoyo para la formación en la ingeniería de sistemas espaciales, usando un CubeSat Bus basado en componentes *Commercial off-the-shelf* previamente diseñado.

Objetivos específicos:

1. Diseñar subsistemas y verificar su funcionamiento mediante prototipo con *Protoboard.*
2. Fabricar cada subsistema del CubeSat Bus, integrarlos y verificar el cumplimiento de los requisitos, usando como referencia la propuesta de interfaz eléctrica de UNISEC para CubeSats, *CubeSat Subsystem Interface Definition* (CSID) (Busch & Schilling, 2016).
3. Diseñar una experiencia educacional en base a las capacidades del CubeSat Bus fabricado, tomando como referencia experiencias educativas de la literatura y la propuesta de trabajo de EYAS-Sat (Jerry Sellers, 2011).
4. Validar el CubeSat Bus, poniéndolo a prueba en el concepto de operaciones definido por un entorno educacional de aula de clases.

1.4 Hipótesis

Este trabajo busca cumplir los objetivos descritos partiendo de la base de que existen en la actualidad satélites de entrenamiento como herramientas de apoyo para la formación en ingeniería de sistemas satelitales. Algunos de estos satélites de entrenamiento han sido desarrollados de forma comercial

como el EYAS-Sat y el E-Sat, pero también hay otros que fueron desarrollados dentro de un programa de transferencia tecnológica como lo es el HEPTA-Sat (Yamazaki, 2018a), o los CubeSat desarrollados y operados por el instituto tecnológico de Kyushu (Kyutech) en el programa BIRDS (Kim et al., 2021; Kumar et al., 2018), además están los fabricantes de CubeSat como tal, que ofrecen un satélite funcional completo, los cuales están diseñados para que puedan integrar cargas útiles según la necesidad del comprador y llevar a cabo misiones en órbita.

El principal inconveniente de los ejemplos mencionados radica en el costo, ya sean herramientas de entrenamiento o satélites funcionales para misiones en órbita, por ejemplo, el EYAS-Sat y E-Sat tienen un costo superior a los 6,000 dólares (CubeSatShop, 2022a) y son netamente herramientas de aprendizaje, no diseñadas para ir a orbita, mientras que CubeSats, sumando los costos de integración, pruebas y lanzamiento llegó a costar hasta $870,000 (Ochocientos setenta mil dólares) por kilogramo en 2011 para una determinada misión (Woellert et al., 2011), cifra que se ha reducido en los últimos años por la reducción de los costos de lanzamiento gracias a los vehículos reutilizables (Pilcher, 2021), y que eventualmente con la evolución de los componentes COTS podría reducirse aún más al no requerir de materiales ni componentes con un TRL (*Technology Readiness Level*) muy elevado, el cual es un parámetro que indica confiabilidad de los componentes en ingeniería aeroespacial (NASA, 2020), a medida que este parámetro aumenta, significa que el ambiente donde se ha validado el componente es más relevante y cercano al ambiente de operaciones de diseño propiamente tal, respecto a un TRL menor.

Así, este satélite de entrenamiento deberá estar compuesto por los subsistemas previstos en el diseño presentado en el Proyecto de Ingeniería Aeroespacial de ahora en adelante PIA (Villarroel, 2022), los cuales se presentan en la Figura 1-11, las funciones de este satélite de entrenamiento están explicitadas en la Figura 1-10, estas funcionas dan lugar a los subsistemas descritos. Otro aspecto del HAISE-Sat es estar integrado mayoritariamente por componentes COTS adquiridos en el mercado y satisfaciendo las condiciones de diseño presentadas más abajo.

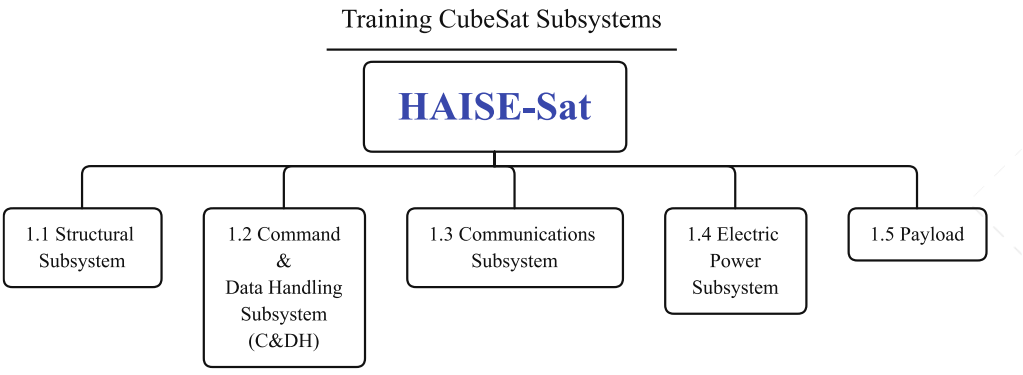


Figura 1-11. Subsistemas del HAISE-Sat obtenidos del diseño realizado en el PIA.

Especificando los requisitos y restricciones que debe cumplir el CubeSat Bus de entrenamiento para entrenamiento académico a fabricar se obtienen las siguientes condiciones de diseño:

1.4.1 Condiciones de diseño del CubeSat Bus:

Este trabajo, continua la línea de trabajo realizada en el PIA, compartiendo con este restricciones y requisitos que guiarán el desarrollo del satélite de entrenamiento a integrar, cada restricción y requisito cuenta con un identificador: C.# corresponde a Restricciones por *Constraints*, mientras que R.# corresponde a Requisitos por *Requirements.*

Restricciones:

- C.1 La fabricación del CubeSat Bus está delimitado por las restricciones de volumen, masa y forma del estándar CubeSat definido por el CubeSat Design Specification Rev. 14 (Cal Poly, 2020).
- C.2 El costo total de fabricación del CubeSat Bus debe ser inferior a 500 dólares estadounidenses.
- C.3 Los procesos de fabricación requeridos deben estar disponibles en la Facultad de Ingeniería.
- C.4 Cualquier cambio al diseño, debe satisfacer las restricciones iniciales de diseño explicitadas en el Proyecto de Ingeniería Aeroespacial (Villarroel, 2022).

Requisitos:

- R.1 El CubeSat Bus debe ser operable en la sala de clases.
- R.2 El CubeSat Bus debe seguir una lógica de integración usando una placa de conexiones perpendicular a las placas de los subsistemas, similar a la vista en el *CubeSat Subsystem Interface Definition* propuesto por UNISEC (Klaus Schilling & Stephan Busch, 2017).
- R.3 La interfaz eléctrica del CubeSat Bus debe permitir integrar un subsistema adicional a los subsistemas considerados en el diseño preliminar.
- R.4 El CubeSat Bus debe tener una autonomía energética de 2 horas o más.

En cuanto a las condiciones de diseño de la experiencia de aprendizaje, están ligadas al trabajo previo (PIA) y queda analizar las características que debe poseer esta experiencia *Hands-on-learning* basado en la literatura.

1.5 Metodología de Trabajo

Para suplir la necesidad de contar con una herramienta de aprendizaje de bajo costo (delimitado por el presupuesto indicado en las restricciones de diseño), se fabricará una, utilizando componentes disponibles en el mercado que, si bien no están diseñados para operar en el espacio, cada uno de ellos

es capaz de simular de forma relevante el comportamiento esperado de los componentes que integran un satélite real. Esto se lleva a cabo integrando el CubeSat Bus de entrenamiento con componentes *Commercial Off-The-Shelf* siguiendo la misma arquitectura funcional de un satélite real, con tal de que la experiencia de aprendizaje realizada con esta herramienta sea símil a un CubeSat diseñado para órbita.

Para ello se prevén los siguientes métodos para apuntar a la realización de los objetivos declarados:

1.5.1 Métodos de verificación de requisitos presentados *Systems Engineering Body of Knowledge* SEBoK

Una forma de establecer si la herramienta cumple con lo indicado en las condiciones de diseño es realizar procesos de verificación y validación de requisitos, a continuación, se presentan métodos para realizar esto, basados en las recomendaciones dadas en la guía de SEBoK (INCOSE, 2020):

- **Análisis:** Técnica basada en evidencia analítica obtenida sin intervención del elemento, usando modelación matemática o estadística, razonamiento lógico, modelando y simulando bajo condiciones definidas para denotar el cumplimiento teórico de los requisitos. Principalmente usado cuando el ensayo en condiciones reales de operación es prohibitivo en costo o calendarización o no puede ser conseguida, además de lo repetible que deba ser el ensayo para entregar buenos resultados.

- **Analogía o similitud:** Técnica basada en evidencia de elementos similares al que se busca verificar. Debe demostrarse que el contexto en el que se verificó el elemento con el cual se compara es extrapolable al elemento por verificar por igualdad de condiciones en el ambiente en el que se busca verificar.

- **Inspección:** Técnica basada en examinación visual o dimensional de un elemento, depende del sentido de la vista humano o de métodos de medición y manipulación. Generalmente es no destructivo y no requiere ensayos. Usado comúnmente para revisar propiedades como color, peso, documentación etc.

- **Demostración:** Técnica usada para demostrar la operación de un elemento especifico en un ambiente operacional y observable, sin la necesidad de realizar mediciones exhaustivas. Las observaciones son comparadas con las respuestas esperadas que debiese tener el elemento. Usualmente utilizado cuando el comportamiento del elemento radica en un fenómeno probabilístico o estadístico.

- **Ensayo:** Técnica llevada a cabo en el elemento a verificar donde se registran capacidades funcionales, características medibles, operabilidad, soportabilidad del elemento o desempeño siendo cuantificables y comparadas con experimentación previa o simulaciones. Puede requerir equipamiento sofisticado para llevarse a cabo.

- **Muestreo:** Técnica basada en verificación de características usando muestras. Se define un numero de muestras y rangos de tolerancia relevantes, para el contexto del elemento, en los cuales dichas muestras debiesen comportarse.

En este trabajo donde principalmente se busca construir la herramienta basada en un CubeSat, el análisis, uso de analogías, inspección, demostración y ensayo resultan ser técnicas relevantes para asegurar que el CubeSat Bus construido cumple con su propósito como herramienta de aprendizaje, además conocer bien las capacidades del *Hardware* logrado permite establecer cuáles son los puntos de mejora que permitirán escalar su diseño a versiones más refinadas que incorporen propiedades presentes en CubeSat reales.

1.5.2 Diseño detallado de la arquitectura del HAISE-Sat

Para lograr el producto final que será el *Hardware*, hace falta definir los detalles respectivos de cada uno de los subsistemas considerando las interfaces requeridas para lograr la integración de estos, además se debe considerar los procesos de fabricación requeridos para pasar del diseño a la construcción, con tal de tomar en cuenta la disponibilidad de herramientas en la Facultad de Ingeniería donde se desarrollara la construcción de este CubeSat Bus.

Para obtener este diseño previo a la construcción, se usará como referencia la documentación existente sobre satélites de entrenamiento como el HEPTA-Sat (Yamazaki, 2015, 2018b) y el manual de usuario del EYAS-Sat (EyasSat, 2016), además la documentación de UNISEC referente a la definición de una interfaz eléctrica, el CSID (Klaus Schilling & Stephan Busch, 2017) será utilizada para complementar la fase de diseño eléctrico.

1.5.3 Prototipo del HAISE-Sat

Previo a la construcción de la versión completamente integrada del HAISE-Sat con cada uno de los subsistemas establecidos (Figura 1-11), es necesario verificar que la arquitectura de este funcione de forma correcta utilizando la interfaz eléctrica con la que se establecerá la conexión de cada uno de los componentes, para ello se probará utilizando cables removibles entre cada componente e interfaces que faciliten la conexión entre estos como lo es una *Breadboard* o también conocida como *Protoboard.* Haciendo esto se podrá verificar si el diseño detallado de los subsistemas y del HAISE-Sat logrado es apto para ser construido, reduciendo así las probabilidades de falla al momento de integrar el CubeSat Bus, ya que, con frecuencia las fallas en sistemas complejos suelen ocurrir en las interfaces que interconectan a cada uno de los componentes (Ryschkewitsch et al., 2009).

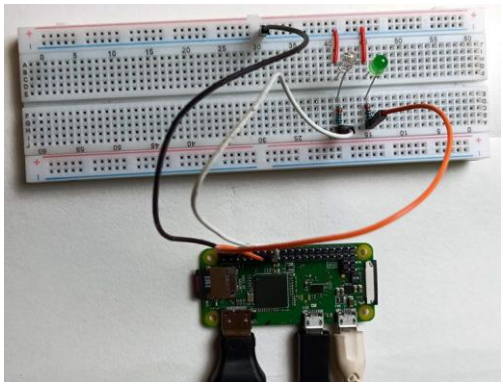


Figura 1-12. Breadboard utilizada para realizar conexiones en una Raspberry Pi Zero, imagen recuperada de (Ashish MIshra, 2021)**.**

1.5.4 Fabricación del HAISE-Sat

En la fase de diseño detallado del HAISE-Sat se definirán los procesos de fabricación requeridos para la elaboración de cada placa de circuito, además de la producción de una estructura que cumpla con el estándar CubeSat, que permita lograr un grado de integración tal, que el HAISE-Sat pueda tratarse como una unidad, facilitando su uso y manipulación.

Para la fabricación de los subsistemas y sus placas de circuitos y la parte estructural del HAISE-Sat, se utilizará software de diseño asistido por computadora y manufactura asistida por computadora, CAD/CAM, como opción disponible existe *Software* gratuito y también *Software* de pago, la Universidad provee de licencias académicas para la utilización de algunos de estos como las herramientas que ofrece AutoCAD y Ansys, con ellos se generarán los recursos requeridos para pasar del diseño a la fabricación e integración de cada subsistema mediante herramientas y maquinaria disponibles en la Facultad de Ingeniería, un precedente es la existencia del Centro para la industria 4.0 que dispone de herramientas que están disponibles para realizar la producción de placas de circuitos electrónicos e impresión 3D.

1.6 Metodología de trabajo para el diseño de la experiencia de aprendizaje.

Es necesario recopilar información relativa a cada uno de los subsistemas para basar su diseño dado que esto influye al momento de la aplicación de esta herramienta en experiencias de desarrollo, integración y operación de satélites en el contexto de los sistemas espaciales, involucrando procesos que se realizan en la academia y la industria con tal de obtener una buena representación del desarrollo de misiones espaciales reales, orientado principalmente en el área de los *Small Satellites* donde se desempeñan los CubeSat. Instituciones relevantes en el desarrollo de *Small Satellites* encontradas en el Proyecto de Ingeniería son CalPoly, Kyutech, INPE y la Universidad de Würzburg, por lo que son un punto de partida desde donde obtener la información necesaria para obtener un diseño final que cumpla con los requisitos indicados en las condiciones de diseño.

CAPITULO 2: Marco teórico

En este trabajo, el objeto de estudio son los satélites, en particular los CubeSat que corresponden a *Small Satellites* descritos en el capítulo introductorio. Un satélite artificial para el contexto de este trabajo, satélite, a grandes rasgos es un dispositivo que orbita la tierra, moviéndose a varios kilómetros por segundo y que lleva a cabo una misión definida, como se mencionó con anterioridad, brindar comunicaciones entre distintas latitudes del planeta, televisión satelital, internet, observación terrestre o llevar a cabo experimentos científicos en ambientes relevantes como lo es un ambiente de microgravedad. Como uno de los objetivos de este trabajo es un integrar una herramienta que simule las características de un CubeSat, es relevante describir la arquitectura de estos satélites ya que esta permite tener una visión general en el diseño de esta herramienta.

2.1 Arquitectura de un Small Satellite.

Una generalización de la arquitectura de los Small Satellite se logra ver en la descomposición funcional de este, Kiselyov en su trabajo realiza una esquematización de esta descomposición funcional o FBS (*Function Breakdown Structure*) vista en la Figura 2-1.

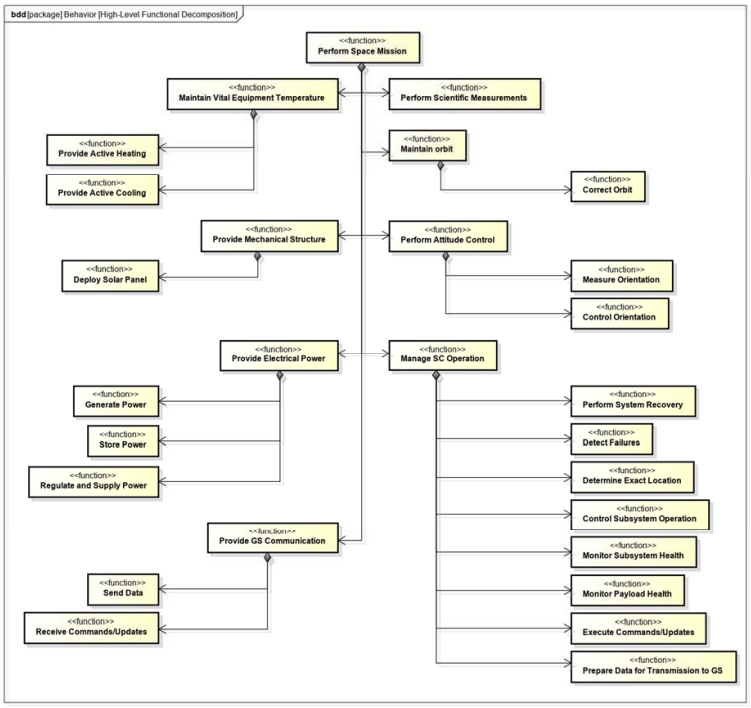


Figura 2-1. Descomposición funcional de un satélite (Kiselyov, 2020)

En el Proyecto de Ingeniería Aeroespacial (Villarroel, 2022) del cual se sustenta este trabajo se realizó la elección de las funciones que eran relevantes para el diseño de una herramienta de aprendizaje a partir de esta descomposición funcional de la Figura 2-1.

La asignación a algo físico de estas funciones se logra generando un *Product Breakdown Structure* (PBS) que jerarquiza según los niveles descritos en la Figura 1-1, a subsistemas, componentes, partes, etc., donde cada una tendrá una función definida, un ejemplo de un PBS se puede ver en la Figura 2-2.

La composición de los satélites se puede dividir en subsistemas según la jerarquía de ECSS (ECSS Secretariat, 2012) y es relevante describir cuales son y como se ven afectados por la misión que debe cumplir el satélite. A continuación, se utiliza la descripción dada por Golkar (Poghosyan & Golkar, 2017) para los subsistemas de un satélite.

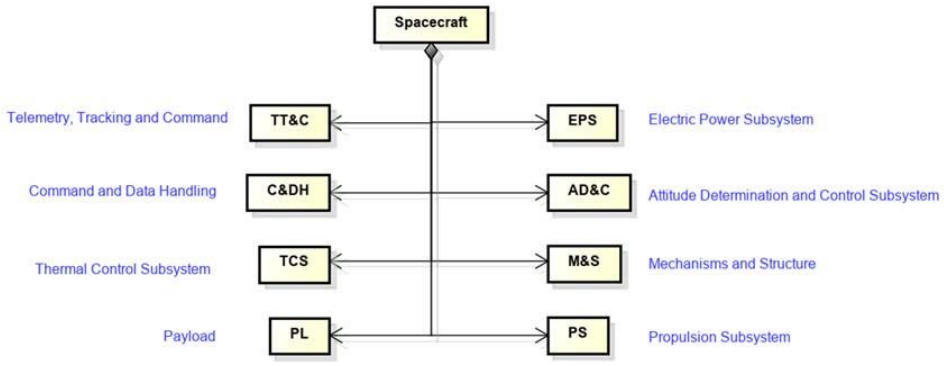


Figura 2-2 Descomposición en productos de un satélite (Kiselyov, 2020).

2.1.1 Subsistema de potencia

En sus siglas en inglés, *Electric Power Subsystem* (EPS), este subsistema se encarga de generar, almacenar y distribuir energía al satélite, además de regularla y controlar la forma en que la entrega. Es relevante en el espacio ya que la principal fuente de energía que permite sostener las operaciones corresponde a la energía solar, y dado que requisitos de la misión se traducen en requisitos energéticos del satélite, características como qué tipo de orbita describe la misión son críticos para el diseño de este subsistema dado el tiempo que el satélite permanecerá a la sombra, eclipsado por la tierra. Por otro lado, los requisitos de desempeño del resto de subsistemas también afectan a la energía que debe suministrar este subsistema de potencia.

2.1.2 Subsistema de comunicaciones

El subsistema de comunicaciones corresponde a la interfaz entre el satélite y la estación en tierra que opera y recibe los datos del satélite, esto permite descargar los datos de la misión, carga útil o *Payload*, y también los datos referentes al estado del satélite. Al contar con este subsistema, es posible recibir comandos e instrucciones desde tierra para operar el comportamiento del satélite, además permite también establecer comunicaciones entre satélites. El método que ha sido ampliamente usado hasta hoy son señales de radio frecuencias, y debido a las grandes distancias que se deben enviar estas señales, la potencia necesaria para ello involucra al subsistema de potencia en gran medida, siendo una vez más el tipo de orbita que requiere la misión un aspecto muy importante para el correcto diseño de este subsistema en paralelo con otros subsistemas.

2.1.3 Subsistema de comandos y manejo de datos

Por sus siglas en inglés, *Command and Data Handling Subsystem*, C&DH, corresponde al subsistema que se encarga de recibir, validar, decodificar y distribuir comandos al resto de subsistemas, a la vez que recibe, prepara y almacena los datos relativos al estado del satélite, y datos propios de la misión para ser usados o para ser descargados en tierra. En términos generales corresponde a la computadora de vuelo del satélite, y que, dependiendo de la misión, se va a requerir tener mayor o menor potencia de cálculo a bordo de la nave, así como mayor capacidad de almacenamiento.

2.1.4 Subsistema estructural

El subsistema estructural corresponde al esqueleto del satélite, es aquel que debe soportar mecánicamente todas las cargas a las cuales se somete este y mantener todo en su lugar, a la vez puede ser aprovechado para proteger al resto de los subsistemas de fenómenos que se dan en órbita como la radiación térmica e ionizante, o basural espacial que pueda comprometer a la misión.

2.1.5 Subsistema de propulsión

El subsistema de propulsión se encarga de dotar al satélite con la capacidad de realizar cambios de orbita (ya sea en altura o características de la órbita), un control de actitud más flexible, mantener formaciones satélites como constelaciones, mantener la órbita en el tiempo dado los requerimientos de la misión, como así también la capacidad de sacar de orbita al satélite bajando su altura para que posteriormente se queme en la atmosfera o permanezca en una zona orbital destinada a satélites inoperativos una vez haya cumplido su ciclo de vida o por fallas irreparables. En la actualidad se

puede observar propulsión química y eléctrica en satélites de diferentes tamaños, pero también hay aquellos sin combustible que aprovechan la radiación solar y la presión que esta ejerce en superficies para propulsar naves espaciales en misiones interplanetarias.

2.1.6 Payload

Este "subsistema" del satélite corresponde a la suma de todos los instrumentos que serían la carga útil que lleva el satélite, desde sensores, cámaras, otros vehículos espaciales, hasta transceptores de comunicaciones y cualquier tipo de instrumentación científica que justifique la razón de estar en órbita del satélite. Si bien existen muchos satélites experimentales, la carga útil en esos casos pasa a ser el subsistema o componente que está siendo puesto a prueba en el espacio. En base al *Payload* se pueden diseñar *Satellite Bus* (todos los subsistemas necesarios para operar al satélite o si se quiere mantener su órbita en el tiempo) o viceversa, para un Satellite Bus existente, se adapta un Payload para integrarlo a este.

2.1.7 Subsistema térmico

Este subsistema se encarga de asegurar la sobrevivencia del satélite a las condiciones adversas del espacio, donde existen altos gradientes de temperatura (originados del casi frío absoluto del espacio con el calentamiento producto de la radiación solar) que pueden comprometer la integridad del satélite si no se compensan. Por otro lado, existen *Payload* que requieren de un rango de temperaturas para operar acorde al comportamiento de diseño, así este subsistema de control térmico se encargará de asegurar los requisitos térmicos del Satellite Bus y del Payload para el correcto funcionamiento de estos, asegurando el cumplimiento de la misión en el tiempo. Se pueden observar controles pasivos, ya sea por aislación térmica y tratamiento de superficies, y activos, con fuentes de calor internas de satélite consumiendo energía, logrando el balance térmico del satélite. Este subsistema es sumamente relevante dado que influye en los requisitos de energía que debe cumplir el satélite según su misión.

2.1.8 Subsistema de orientación, navegación y control

Este subsistema por sus siglas en inglés, *Guidance, Navigation and Control*, GNC, es una combinación entre el *Orbit Determination and Control Subsystem* (ODCS), el subsistema que se encarga de determinar y controlar la posición orbital del satélite en el tiempo, y el *Attitude Determination and Control Subsystem (ADCs)*, subsistema que se encarga determinar y controlar la orientación o actitud del satélite respecto a algún marco de referencia. Dado los requerimientos de la misión, el apuntado del satélite, es decir el ajuste de su actitud puede ser más o menos exigente, también la determinación de orbita puede ser un aspecto critico de la misión ya que se necesita saber

con cierta exactitud donde se encuentra en el espacio para efectuar ciertas operaciones como apuntar una antena por ejemplo, por lo que llevará a tener un GNC más robusto y preciso que una misión menos exigente, aumentando el costo energético y la masa total del satélite, o, por otro lado, puede aumentar los requerimientos de capacidad de cómputo del C&DH al tener que satisfacer una mayor frecuencia con la cual mide y estima su orientación y posición en el tiempo.

2.2 Lógica de integración de los subsistemas

En el PIA (Villarroel, 2022), se determinó que la lógica de integración del CubeSat Bus seguiría lo presentado por UNISEC, en particular se toma el ejemplo de los satélites desarrollados bajo el programa BIRDS (Azami et al., 2019; Maskey et al., 2022). En UNISEC se propone cambiar el conector que se estaba utilizando y el que manejaban los fabricantes de COTS para CubeSats a la fecha, 2017, el PC104, que con todos sus inconvenientes por haber sido diseñado para hardware en tierra, sumado a los diferentes problemas que surgían la tratar de integrar componentes de distintos fabricantes según los resultados que arrojó una encuesta a integradores de CubeSat (Bouwmeester et al., 2017), se concluyó que era necesario un nuevo conector que reuniera ciertas características las cuales están descritas en dicho trabajo.

Además, para facilitar la fase de pruebas un nuevo método para integrar el satélite era necesario, por lo que optaron por usar una placa de conexiones, que sería perpendicular a las placas de los subsistemas, llamada *Backplane Board* o BPB, esta placa tendría un conector estandarizado manteniendo un solo Bus de datos para todos los subsistemas, el cual está documentado en los trabajos llevados a cabo durante el desarrollo de los satélites UWE en la Universidad de Würzburg (Busch & Schilling, 2016), sin embargo en el programa BIRDS, utilizan un conector distinto, menos compacto pero manteniendo la lógica de integración la cual se puede ver en la Figura 2-3.

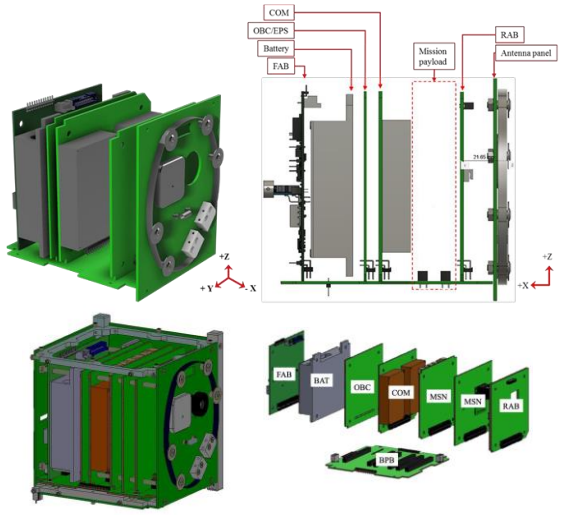


Figura 2-3. Lógica de integración de satélites del programa BIRDS.

Así, para este trabajo se utilizará esta lógica de integración de los subsistemas, ya que facilita el análisis de los subsistemas cuando se realicen actividades durante su uso como herramienta de aprendizaje, ya que como destacan en el trabajo relacionado con los satélites UWE de Würzburg, facilitó la integración, y pruebas de los satélites, así como la mantención, modificación y reemplazo de subsistemas mientras se realizaba el *flat-sat*, el cual es integrar el satélite fuera de la estructura en una superficie plana, teniendo acceso a todos los componentes que integren a los subsistemas cuando se trabaja en la integración del prototipo y del modelo que irá a orbita (Busch & Schilling, 2016).

CAPITULO 3: Diseño y prototipado del HAISE-Sat

Para construir este satélite de entrenamiento, es necesario definir un diseño tanto eléctrico como estructural de cada uno de los subsistemas según corresponda. Los principales desafíos en esta etapa contemplan el uso de dos propuestas de estandarización, para la parte estructural del CubeSat Bus existe el Estándar CubeSat (Cal Poly, 2020) y para la parte eléctrica, la propuesta de interfaz eléctrica de UNISEC (Klaus Schilling & Stephan Busch, 2017). Con estas dos condiciones de diseño (las cuales están incluidas en el diseño preliminar logrado en el Proyecto de Ingeniería Aeroespacial), se establecen los diseños de cada subsistema con un mayor nivel de detalle, complementando y solucionando las dificultades que presentaba el diseño preliminar.

El *Product Breakdown Structure* del diseño preliminar se observa en la Figura 3-1.

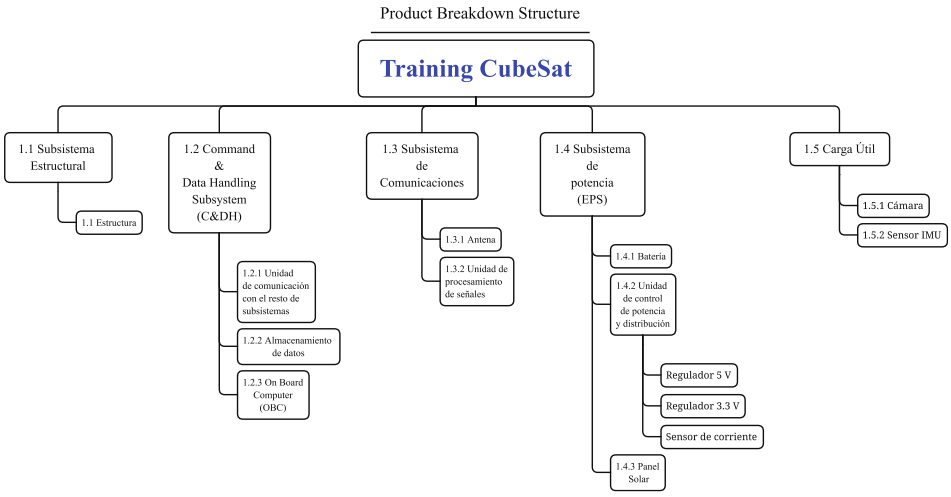


Figura 3-1. *Product Breakdown Structure* del HAISE-Sat en el diseño preliminar

3.1 Software utilizado.

Para toda la fase de diseño se recurrió a software CAD, esto con tal de facilitar la posterior fabricación de los componentes requeridos para la integración del HAISE-Sat. El software de este satélite de entrenamiento son programas en lenguaje Python ejecutados en el sistema operativo de una Raspberry Pi y en un computador con sistema Windows ejecutando un archivo Python que inicia la estación de control.

3.1.1 EasyEDA

Consiste en un software gratuito para diseño asistido por computadora, CAD, de circuitos electrónicos, permite almacenar los proyectos en la nube para poder acceder a ellos en cualquier parte teniendo acceso a internet. Se ejecuta en el navegador por lo que es posible trabajar en este software con computadores de bajos recursos. Posee un editor de diagramas eléctricos y un editor de diseño de circuitos impresos, PCB, permitiendo trabajar de forma paralela entre diagrama y PCB, la interfaz de usuario es bastante intuitiva por lo que su uso no supone mayores complicaciones para usuarios que nunca han realizado diseño de circuitos eléctricos en computadora. La gran ventaja que posee es su inmenso repositorio de librerías con componentes, desde los suministrados por empresas que fabrican un determinado componente, hasta librerías generadas por los usuarios de la comunidad que sustentan a este software gratuito.

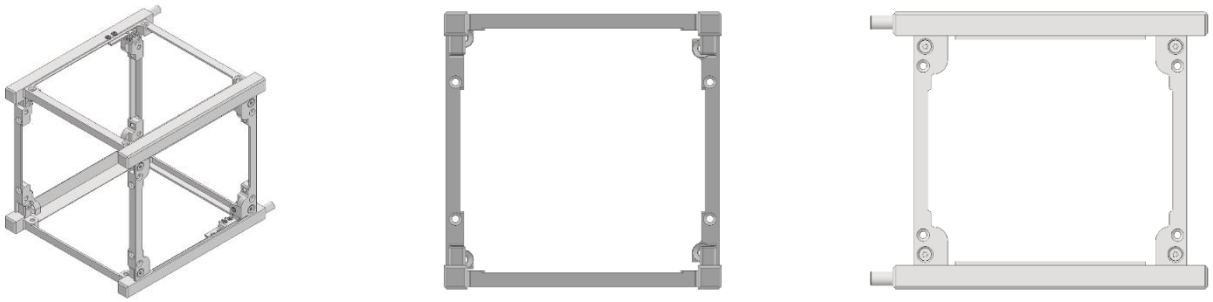
3.1.2 Inventor

Consiste en un software CAD 3D para diseño mecánico, generar su documentación y simular los elementos diseñados según las solicitaciones que uno ingrese al modelo. Tiene funcionalidades como exportar un tipo de archivo que pueda leer EasyEDA y permitir realizar el diseño de las piezas mecánica en paralelo con las placas de circuitos de los subsistemas.

3.2 Subsistema Estructural

Este subsistema tiene la función de soportar las cargas que pueda sufrir el satélite. En el caso de su misión como herramienta de aprendizaje, la estructura del HAISE-Sat debe soportar las cargas al manipularse, esto implica también mantener todo en su sitio, asegurando la conectividad eléctrica de todos los subsistemas, así como la integridad de sus componentes.

La estructura está directamente restringida por el estándar CubeSat, y con lo popular que ha sido este estándar en los últimos 10 años, se pueden encontrar trabajos *open source* de modelos 3D de la estructura de CubeSats. Con esto no hace falta diseñar desde cero la estructura del HAISE-Sat. Dos fuentes relevantes que proveen un modelo 3D para la estructura son ISISPACE, Figura 3-2, compañía holandesa que fabrica y vende subsistemas para CubeSat, y un modelo 3D simplificado del ArduSat, satélite lanzado en 2013 el cual estaba basado en Arduino.



a) Vista isométrica **b) Vista frontal** **c) Vista lateral izquierda**

Figura 3-2. Modelo tridimensional de CubeSat 1U de ISISPACE (ISISPACE, 2022)**, archivo .step visualizado con Inventor de Autodesk.**

El modelo proporcionado por ISISPACE, Figura 3-2, entrega información detallada de las uniones requeridas como tornillos, agujeros e hilos que permiten integrar el resto de los subsistemas del CubeSat, en particular este modelo incluye un *Kill Switch* incorporado a la estructura junto a sus soportes, un *Kill Switch* es un interruptor que se encarga de evitar que el satélite encienda ciertos dispositivos mientras aún está montado en el sistema de despliegue POD y pueda poner en peligro al resto de carga útil del vehículo que los llevará a su órbita (NASA, 2017).

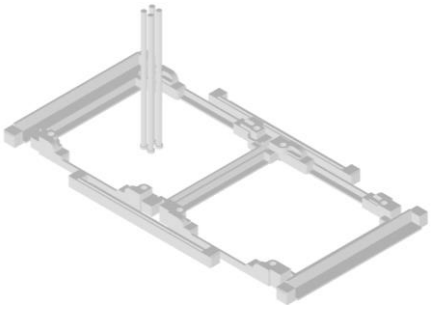
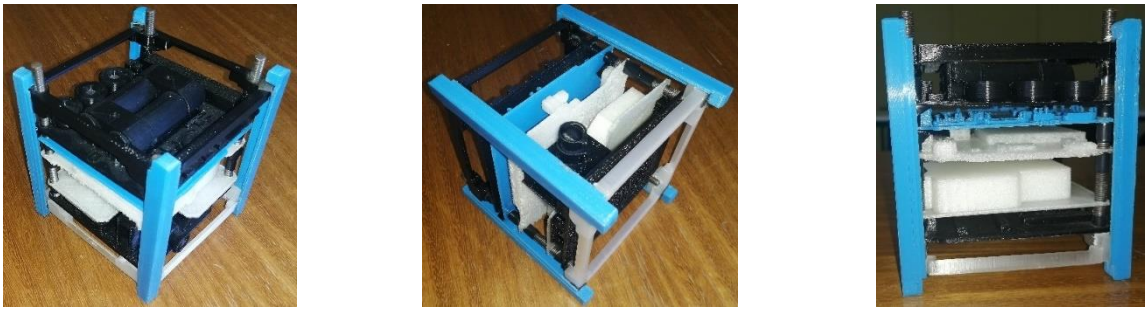


Figura 3-3. Modelo tridimensional original de ArduSat (MAKERFAM, 2012) **, archivo .stl visualizado con Inventor de Autodesk.**

Por otro lado, el modelo proporcionado por el proyecto ArduSat, Figura 3-3, se caracteriza por presentar una versión simplificada de la estructura, el archivo que entregan resulta con todas las piezas situadas en un mismo plano con tal de facilitar la fabricación de esta con impresión 3D.

Un ejemplo ya impreso del ArduSat se observa en la Figura 3-4.

Un aspecto importante de la estructura es que define el tamaño máximo de las placas de circuitos de cada subsistema y la forma en que se fijaran estas a la estructura con tal de asegurar la conexión eléctrica de cada subsistema al ser manipulado. En los archivos que entregan con el proyecto de ArduSat (MAKERFAM, 2012) está incluida una virtualización de las placas de circuitos que fueron incorporadas en la versión final que fue a órbita, por lo que son útiles para establecer que dimensiones deben tener las placas de los subsistemas en este trabajo, con tal de compatibilizar la estructura con el diseño de las placas de circuito.



a) **b)** **d)**

Figura 3-4. ArduSat impreso en la facultad, con una representación de sus subsistemas.

3.2.1 Primera iteración del diseño estructural

Utilizando como base la estructura del ArduSat (MAKERFAM, 2012), se importó al ambiente de trabajo de Inventor su archivo 3D, partiendo del modelo indicado en la Figura 3-3 se obtiene la estructura de la Figura 3-5 la cual solo contiene las bases, los largueros serán reemplazados por barras roscadas y las fijaciones laterales se ubican en un ambiente de trabajo nuevo, resultando así las piezas de la Figura 3-6 con tal de facilitar la impresión en caso de fallas durante esta, dejando así la pieza más compleja aislada la que resulta ser las bases.

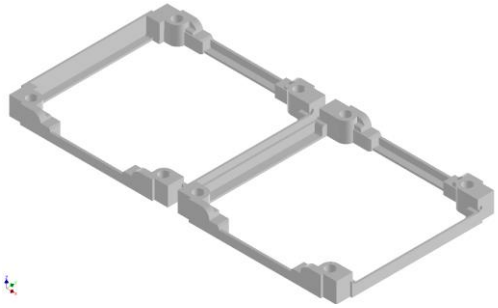


Figura 3-5. Bases de la estructura del ArduSat

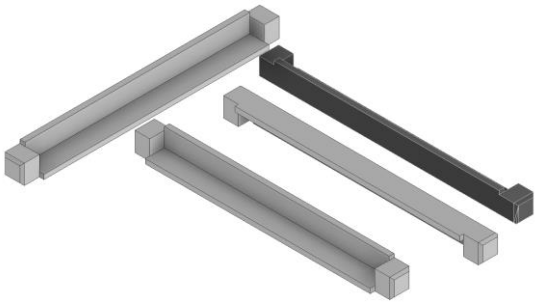
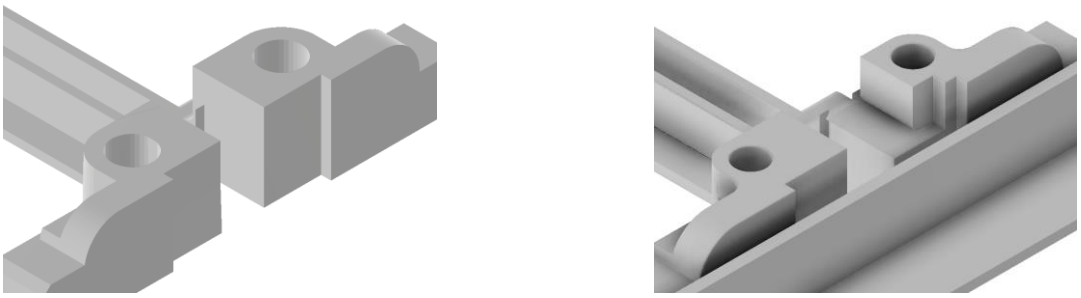


Figura 3-6. Fijaciones laterales de la estructura.

Adicionalmente se modificó el diámetro de los agujeros de la base, el diámetro original era de 4 mm y se aumentó a 5.7 mm para utilizar una barra roscada o espárrago, del cual ya se disponía. Además, se realizaron una serie de modificaciones para rigidizar esta estructura cerca de los agujeros y reducir posibles problemas con el proceso de fabricación cuando se generan paredes muy delgadas. Otra razón para aumentar el diámetro de los agujeros se relaciona con las tolerancias que se logran con este proceso de manufactura aditiva, dando margen para que los agujeros en la pieza fabricada tengan una dimensión menor debido a las características de la impresión como puede ser la expansión del material utilizado.

Estos cambios ya están presentes en la Figura 3-5 y se puede comparar algunas diferencias en la Figura 3-7, se observa que se retiran las fijaciones laterales del modelo las cuales serán trabajadas en un modelo aparte, y en la región de los agujeros se extienden las caras hasta el contorno externo que delimita la zona de integración son los fijaciones laterales, esto se repite en cada agujero con tal de rigidizar la estructura.



a) Versión modificada de la estructura **b) Modelo original**

Figura 3-7. Comparación visual de modelo modificado y modelo original.

La estructura fabricada a partir de este diseño se observa en la Figura 3-8, verificando que mantiene una de las dimensiones establecidas por el estándar CubeSat con un pequeño margen de error, de igual forma de verifica para el resto de las dimensiones un error inferior al 2%.

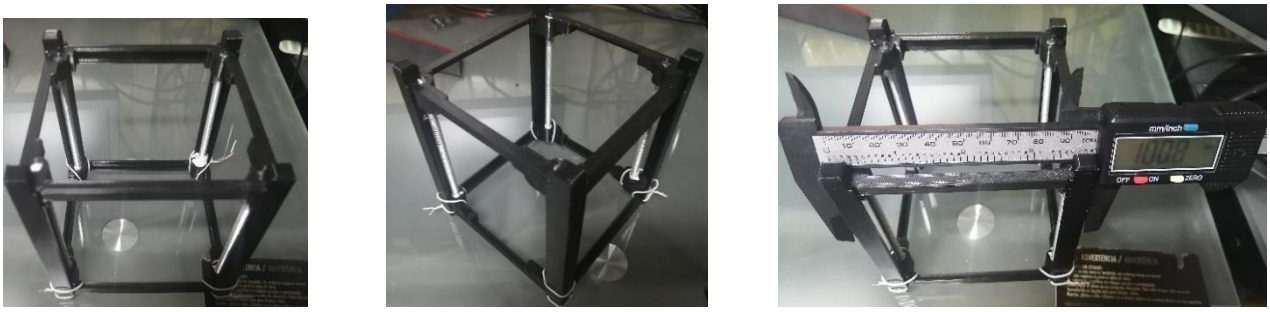


Figura 3-8. Primer diseño fabricado de la estructura.

3.2.2 Segunda iteración del diseño estructural

Con el primer diseño integrado, se encontraron algunos inconvenientes para el armado de la estructura y resultó que no había ningún sistema de sujeción para las fijaciones laterales, esto se evidencia con el uso de alambres para sujetar la estructura en la Figura 3-8, por lo que era necesario generar un nuevo diseño que permitiera fijar cada una de las piezas desde la misma estructura.

Así, se genera un nuevo modelo 3D a partir del original del ArduSat manteniendo las dimensiones externas de la estructura y la ubicación de los espárragos. Para mantener todo unido, una opción resultó ser uniones por interferencia, y al trabajar esa idea se llegó al resultado visto en la Figura 3-9. El concepto es usar ranuras en la estructura para acoplar las piezas a presión, a su vez limitan los grados de libertad traslacionales que tendrán las sujeciones laterales de 3 a 2, las cuales corresponden al plano de la ranura.

Figura 3-9. Unión por interferencia entre fijaciones laterales y bases de la estructura, dimensiones en mm.

Para la interferencia se trabajó en el diseño con una diferencia inicial de 0.05 mm entre la pieza que se inserta y la ranura, siendo la ranura la pieza con mayor espesor en la unión, esto debido a efectos de expansión del material y tolerancias que manejan las máquinas de impresión 3D para luego permitir un ajuste por postprocesado de las piezas mediante remoción de material, las dimensiones de diseño para la unión se pueden ajustar categorizando las tolerancias que maneje la impresora 3D con la que se vaya a trabajar, con tal de reducir el postprocesado.

Un detalle adicional al diseño fue agregar otro elemento que limitara los grados de libertad traslacionales a solo la dirección en que se inserta la pieza, esto se logró mediante la implementación de una segunda aleta en las fijaciones laterales, visto en la Figura 3-10.

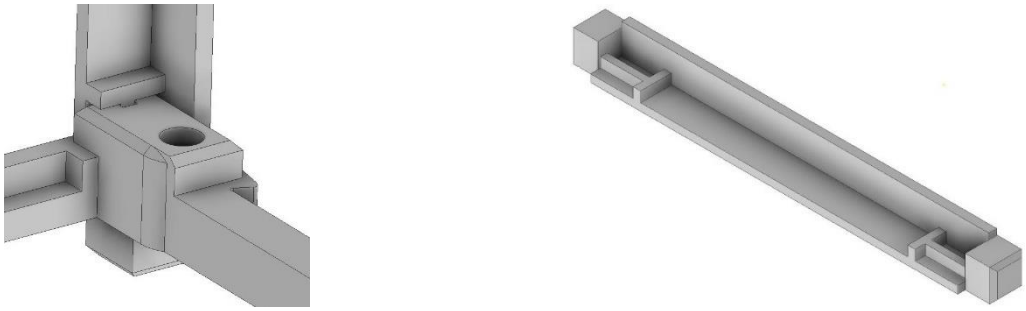


Figura 3-10. Modificación a fijación lateral.

Luego de realizar el postprocesado de las piezas, la cual resultó en remover material en las ranuras de las bases solamente, se realiza la primera fabricación de esta pieza y se puede ver integrada en la Figura 3-11.

La Figura 3-11.a muestra la integración de la estructura con los espárragos y los anillos mientras que en la Figura 3-11.b se ve solamente las bases y las fijaciones laterales manteniendo su posición gracias a las uniones por interferencia. Al agregar los espárragos con los anillos se puede mejorar aún más la unión de las fijaciones laterales con la base, gracias al avance de los anillos que seguirán el hilo, aumentando la presión de contacto entre las terminaciones de los fijadores laterales y la cara externa de las bases, lo cual de implementarse de mejor manera haría que la modificación de la Figura 3-10 no fuera necesaria.

a) **b)**

Figura 3-11. Integración de la segunda iteración del diseño estructural.

Se adiciona un método para integrar los paneles solares previstos en el diseño preliminar del CubeSat de entrenamiento, igualmente se ideó una unión por interferencia usando las dimensiones de los paneles solares adquiridos, el diseño de la unión se observa en la Figura 3-12, la cual es generada al aproximar el recorrido que debiese tener el panel solar para caer en la ranura.

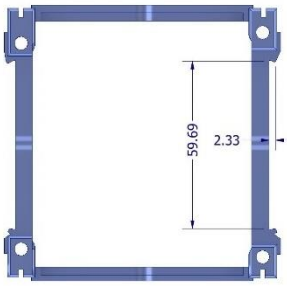


Figura 3-12. Acople para integrar un panel solar a la estructura.

La pieza fabricada en la Figura 3-11 ya posee estas modificaciones y la estructura con el panel solar integrado se observa en la Figura 3-13, donde está fijado solamente por la ranura de la estructura.

Figura 3-13. Integración del panel solar a la estructura.

Los anillos impresos en PLA vistos en la Figura 3-11 serán reemplazados por tuercas para aumentar la presión que generan sobre la estructura, ya que, al ser fabricados en plástico, se desgastan con rapidez en contacto con el esparrago de acero.

3.2.3 Obtención de Huella de subsistemas del ArduSat.

Para el diseño del resto de subsistemas, es necesario que el tamaño de estos sea coherente con el volumen permitido por la estructura, para ello se utilizaron los modelos que entrega ArduSat con tal de compatibilizar la estructura con las placas de circuitos. Se importó a Inventor de Autodesk el modelo que representa el subsistema de potencia del ArduSat, con tal de obtener las dimensiones y la ubicación de los agujeros para los pasadores, el EPS del ArduSat se puede ver en la Figura 3-14, así se obtuvo una huella que permite alocar los agujeros que serán fijados por la barra metálica o esparrago de la Figura 3-8, además de definir las dimensiones que deben tener las placas de circuito, se comparó además con la ubicación de los agujeros en las bases de la estructura y se comprobó que eran idénticas.

Inventor permite exportar los *Sketch* a un archivo con extensión ".dxf", estos archivos pueden ser importados en el software EasyEDA donde se diseñarán las placas y trabajar sobre una geometría para ubicar todos los componentes dentro del espacio del que se dispone. El sketch con la huella se ve en la Figura 3-15.

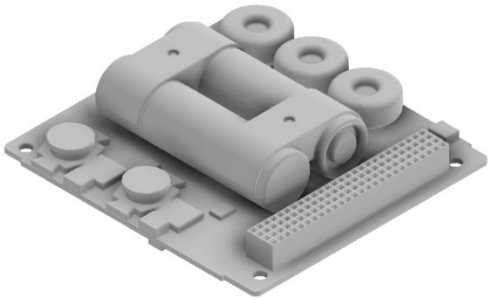


Figura 3-14. Modelo 3D del subsistema de potencia del ArduSat

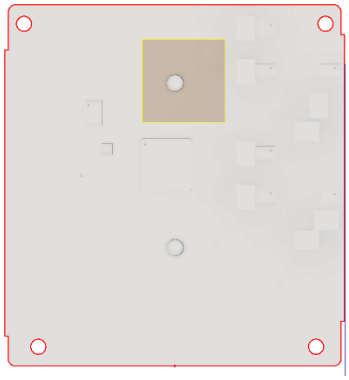


Figura 3-15. Proyección del contorno y agujeros del subsistema de potencia del ArduSat.

Luego de obtener esta huella se importa en EasyEDA el archivo ".dxf " y se obtiene la geometría para poder trabajar desde el editor de circuitos impresos de este software, permitiendo diseñar las placas contenidas dentro del espacio que está disponible, el cual es restringido por la estructura del satélite, se puede ver la geometría importada en el espacio de trabajo de EasyEDA en la Figura 3-16, de esta geometría se basará el diseño de las placas de circuito .

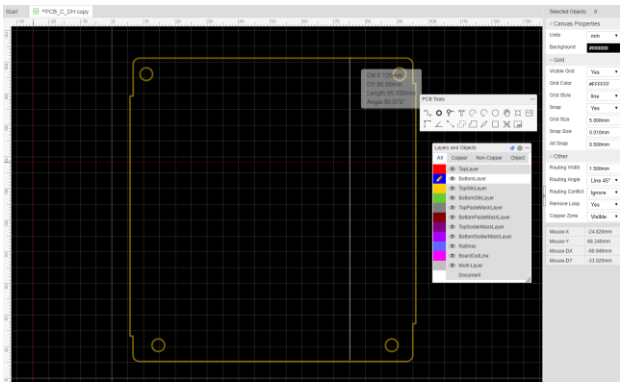


Figura 3-16. Geometría exportada a EasyEDA para diseño de placas de circuito.

3.3 Asignación de pines al conector eléctrico.

Del PIA (Villarroel, 2022) se legó la asignación de pines mínima que debe tener el conector. Este satélite de entrenamiento contará con un conector de 50 pines, 2 filas de 25 pines cada una, el cual permitirá conectar los subsistemas a una placa de distribución de forma perpendicular, similar a como se aprecia en la Figura 3-17, donde la placa de distribución lleva el nombre de *Backplane* en la nomenclatura usada en los trabajos del programa BIRDS (Azami et al., 2019; Kim et al., 2021; Kumar et al., 2018).

Los conectores para cada subsistema que lo requiera tendrán la misma asignación de pines con tal de estandarizar el diseño eléctrico del HAISE-Sat para facilitar su entendimiento y para futuras actualizaciones a su arquitectura.

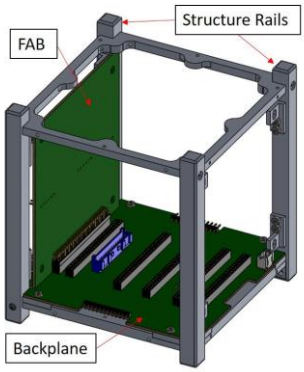


Figura 3-17. Integración de placas de subsistemas en satélites BIRDS (Kumar et al., 2018).

Este conector de 50 pines debe contar como mínimo con las señales indicadas en la Tabla 3-1 para la comunicación y distribución de energía.

Tabla 3-1. Pines con señal asociadas comunes para HAISE-Sat basado en Proyecto de ingeniería.

Señal asociada a un pin	**Descripción**
I^2C SDA	Protocolo I^2C para comunicación con sensores
I^2C SCL	
5.0 V	Señal de 5.0 V regulada
3.3 V	Señal de 3.3 V regulada
GND	Conexión a tierra común para señales digitales

Para darle flexibilidad al hecho de que el HAISE-Sat está basado en Raspberry Pi como computador a bordo, la cual es una decisión de diseño tomada en el PIA (Villarroel, 2022), se asignan los pines de este *Single Board Computer* [2] (SBC) al conector utilizado. Las señales que maneja una Raspberry Pi se ven en la Figura 3-18.

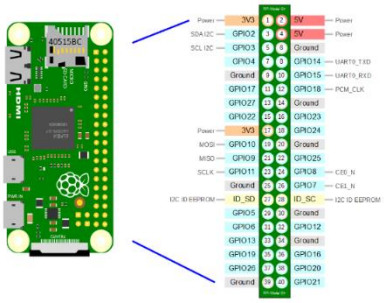


Figura 3-18. Asignación de señales en los pines de la Raspberry Pi Zero (Raspberry Pi Foundation, 2018b)

Asemejando el orden de los pines de la Raspberry Pi y asignando las señales indicadas en la Tabla 3-1 se define la asignación de señales para el conector eléctrico del HAISE-Sat en la Figura 3-19.

Con las señales ya asignadas se puede realizar el diagrama eléctrico de los subsistemas. La particularidad del protocolo I2C empleado en este conector, es que se pueden conectar varios dispositivos en paralelo en las mismas líneas SDA y SCL ya que cada dispositivo tiene un identificador que lo distingue del resto, utilizando así solo dos pines del conector para integrar cualquier dispositivo que utilice el protocolo de comunicaciones I2C para transmitir datos.

En la Tabla 3-2 se resume la función de cada pin del conector para el HAISE-Sat de la Figura 3-19:

[2] SBC son computadores integrados en una sola placa de circuitos conteniendo todo lo necesario para funcionar salvo la fuente de energía que es suministrada de forma externa.

Figura 3-19. Distribución de pines para el conector del HAISE-Sat

Tabla 3-2. Asignación de pines

Señal asociada a un pin	**Descripción**
I^2C SDA	Protocolo I^2C para comunicación con sensores
I^2C SCL	
5.0 V	Señal de 5.0 V DC regulada
3.3 V	Señal de 3.3 V DC regulada
GND	Conexión a tierra común para todo el satélite.
GPIO#	*General Purpose Input/Output*, son señales que puede manejar Raspberry Pi, con tal de que puedan ser aprovechadas en versiones futuras del HAISE-Sat
No	Pin sin señal eléctrica, será retirado del conector macho y en el conector hembra será tapado, con tal de darle Polaridad al conector, evitando conexiones invertidas que puedan dañar a los componentes.

3.4 Subsistema de comandos y manejo de datos (C&DH).

Este subsistema estará basado en el SBC Raspberry Pi Zero y su versión con WiFi, Zero W. Este computador cuenta con la interfaz de comunicaciones I2C y UART los cuales permiten conectar distintos módulos disponibles en el mercado, estos módulos pueden ser sensores, tarjetas de expansión de almacenamiento o sistemas de comunicaciones como radiofrecuencia.

Lo relevante de este computador, es que puede manejar distintos lenguajes de programación de forma nativa como C++ y Python, este último fue utilizado para programar el *Software* de vuelo del HAISE-Sat, además los distintos sensores que se utilizarán además de la carga útil pueden ser operados desde códigos Python, esto permitirá que, mediante comandos, se pueda acceder a lo que está midiendo cada sensor, para luego ser enviado al usuario mediante el subsistema de comunicaciones.

Figura 3-20. Aspecto de la Raspberry Pi Zero (Raspberry Pi Foundation, 2018a).

3.4.1 Primera iteración del diseño eléctrico.

Dada las capacidades de fabricación con las que cuenta el C4i, Centro para la industria 4.0 presente en la universidad, es posible fabricar placas de circuito de doble capa, por lo que está contemplado como restricción de diseño. Utilizando el software libre EasyEDA, se llega a un primer diseño visto en la Figura 3-21.

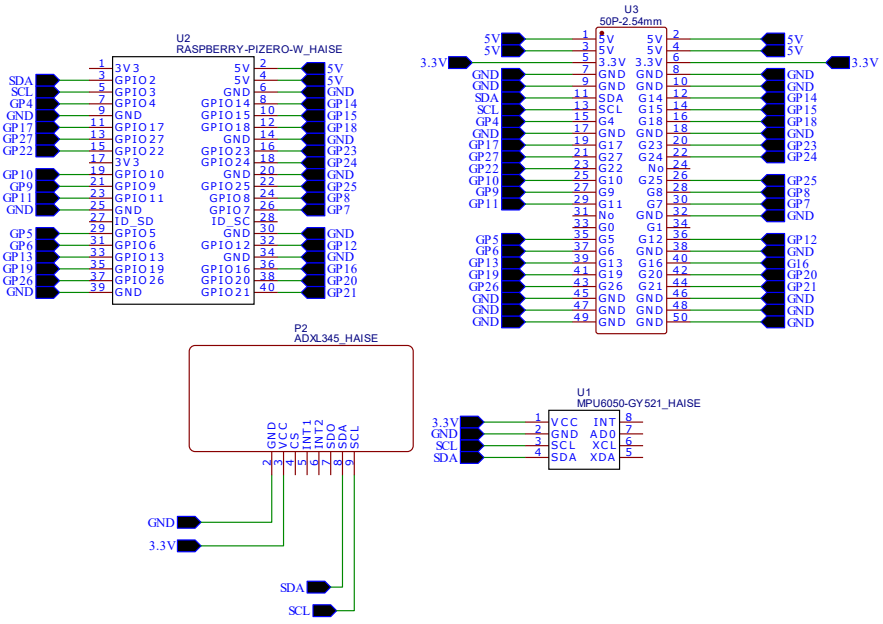


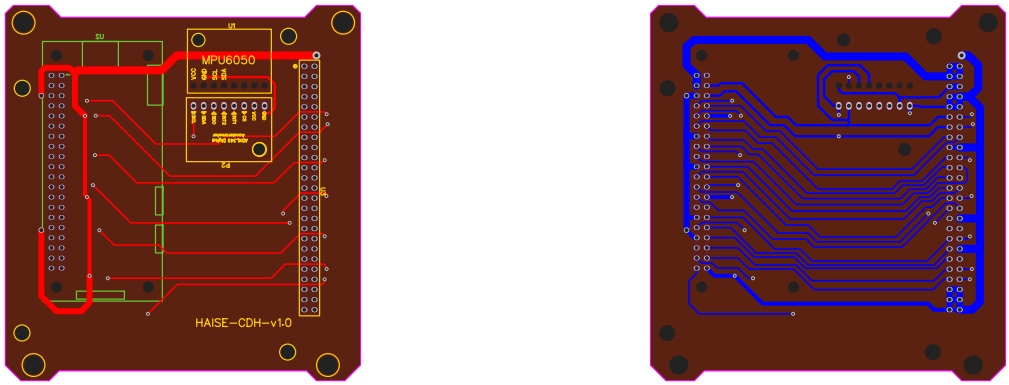
Figura 3-21. Primera iteración diagrama eléctrico para el subsistema de comandos y manejo de datos C&DH.

El bloque que representa las conexiones de la Raspberry Pi Zero está indicado con su etiqueta mientras que el conector utilizado, mencionado en el punto 3.3, está indicado por la etiqueta 50P-2.54mm.

Por otro lado, en esta primera iteración del HAISE-Sat, se opta por incluir en la placa destinada al C&DH, los módulos de acelerómetros, el ADXL345 y giroscopio MPU6050, ya que la placa dispone del espacio para integrarlos y aprovechar el espacio, estos actuaran como *Payload*.

Otro componente del Payload es la cámara y requiere estar directamente conectado a la Raspberry Pi, esta cámara dispone de un conector propio que comunica a esta con el SBC y no está contemplado en el diagrama eléctrico presentado.

La visualización de este diagrama eléctrico en el diseño de la placa de circuitos se puede ver en la Figura 3-22. Además, se sitúan agujeros adicionales a los de la estructura con tal de permitir la fijación usando un soporte para los componentes que queden en voladizo, el cual fue diseñado en paralelo con la placa. El tamaño de la placa de circuitos fue construido a partir de la huella obtenida en el punto 3.2.3 para que sea compatible con la ubicación de los espárragos y los limites interiores de la estructura.



a) **b)**

Figura 3-22. Placa de circuitos del C&DH, vista desde la cara superior. a) Capa superior, b) Capa inferior.

A través del Centro para la industria C4i, se fabrica la placa de circuitos del subsistema C&DH logrando el producto de la Figura 3-23

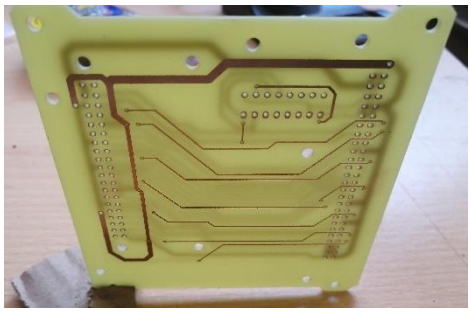


Figura 3-23. Placa de circuitos del C&DH fabricada.

3.4.2 Soporte para componentes del subsistema C&DH.

Dada la forma de integración de los componentes en las placas de circuito para facilitar su manipulación y reemplazo de los componentes, se escogió usar conectores similares a los utilizados en la conexión del BP y subsistemas, esto se observa en un prototipado de las conexiones en la Figura 3-24. Para evitar que los componentes queden en voladizo, hace falta incorporar una estructura que brinde soporte a estos, con tal de asegurar la integridad de las conexiones y de los componentes.

La primera consideración para esta pieza de apoyo es la distancia relativa entre una de las caras de la Raspberry y la cara que enfrenta de la placa de circuitos.

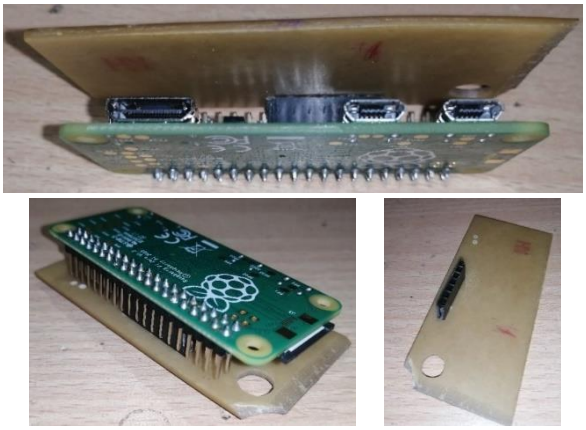
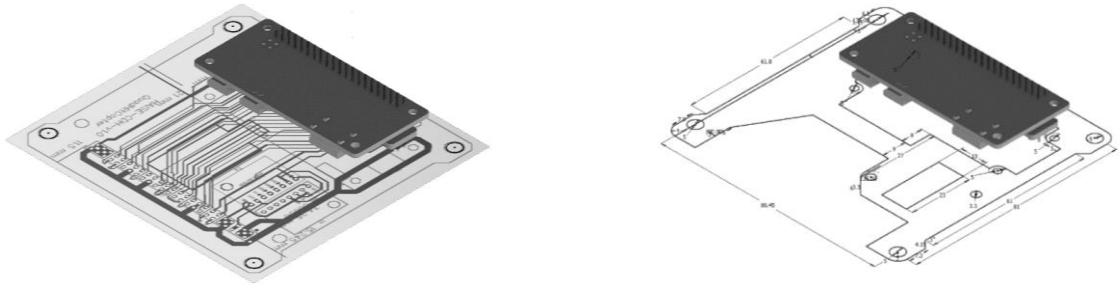


Figura 3-24. Prototipado del montaje de los componentes del subsistema C&DH.

En la Figura 3-25 se esquematiza la dimensión que debe satisfacer el apoyo de los componentes usando un modelo 3D de la Raspberry y los conectores en Inventor, esta dimensión está dada por las dimensiones reales del conjunto conector-Raspberry visto en la Figura 3-24. El modelo de la Raspberry Pi Zero se obtuvo en el sitio web ThingiVerse al igual que el modelo del ArduSat, mientras que el conector se obtuvo de la librería de componentes del distribuidor mayorista de componentes eléctricos Mouser Electronics.

Figura 3-25. Dimensionamiento del soporte para los componentes del subsistema C&DH usando un modelo 3D de una Raspberry Pi Zero.

Con esto, y utilizando la huella obtenida en el punto 3.2.3, se puede diseñar una estructura que sirva de soporte a los componentes electrónicos sin obstaculizar el apoyo con los mismos componentes, haciendo contacto solo en las zonas libres de estos. Para ello se exportó una imagen de la placa de circuitos desde EasyEDA y se exportó a Inventor para hacer el modelado de esta estructura y que coincida con los agujeros de paso de los espárragos, agujeros para fijación y la posición de los componentes. Esto se observa en la Figura 3-26.

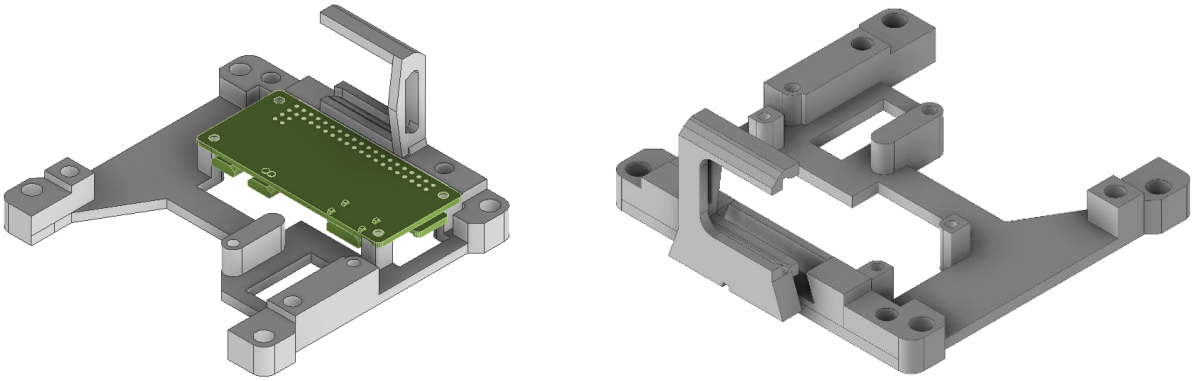


a) **b)**

Figura 3-26. Generación de Sketch para diseño de apoyo de componentes del subsistema C&DH.

De esta forma se define el contorno que tendrá el soporte y se puede generar un sólido a partir de ello que haga contacto con los componentes, adicional se agregan los agujeros para las uniones apernadas.

Luego dándole forma al apoyo hasta la altura que se requiere se obtuvo el modelo de la Figura 3-27, adicional se situaron los agujeros de fijación en las zonas libres que permitía el diseño de la placa de circuitos.



a) b)

Figura 3-27. Modelo 3D del soporte para los componentes del subsistema C&DH.

Además, se diseñó en paralelo un soporte para la cámara con tal de mantener su posición fija mientras el HAISE-Sat realice su misión. Para facilitar la fabricación de este soporte, se generaron dos piezas, vistas en la Figura 3-27.b, el soporte de la cámara y el soporte de los componentes que va unido a la placa de circuitos mediante unión roscada.

El soporte de la cámara se diseñó usando las dimensiones de esta, fijando la cámara en una ranura que restringe su desplazamiento, luego esta pieza es integrada al soporte de la placa mediante una unión por interferencia, las dos piezas unidas se pueden ver en la Figura 3-27.a .

3.4.3 Integración del subsistema C&DH.

Con la placa de circuitos y el soporte fabricados, se integró el subsistema de comandos y manejo de datos C&DH, junto al Payload del satélite de entrenamiento que consiste en la cámara y dos sensores de movimiento inercial, el ADXL345 y el MPU6050. Se utilizó un perno M3 con su respectiva tuerca moleteada para fijar el soporte con la placa de circuitos. El resultado final se puede ver en la Figura 3-28.

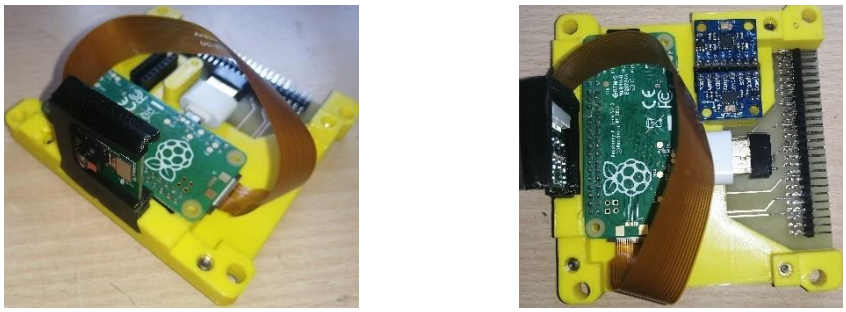


Figura 3-28. Integración del subsistema C&DH.

Cabe destacar que la integración involucra 3 subsistemas, C&DH, Payload y Comunicaciones, esto se debe a que tanto la cámara como el adaptador WiFi para el Raspberry Pi Zero, y el WiFi integrado en la misma Raspberry Pi Zero W, no se pueden ni vale la pena separarlos a una placa aparte, por lo que esta placa tendrá integrados estos 3 subsistemas.

3.5 Payload

Como se mencionó en el punto anterior, el Payload que consiste en una cámara y dos sensores que medirán aceleración y aceleración angular, están integrados en la misma placa que contiene al subsistema de comandos y manejo de datos C&DH. La adquisición de los datos del Payload se realiza mediante códigos Python utilizando el canal de comunicaciones I2C, por lo que permite automatizar la obtención de datos para ser enviados al usuario. Hay que destacar que, al contar con los acelerómetros y giroscopio, se pueden implementar métodos de determinación de actitud en trabajos futuros.

Los sensores utilizados son un ADXL345 que solo cuenta con acelerómetro, y un MPU6050 o conocido como GY-521 que combina acelerómetro y giroscopio en un mismo módulo. Estos sensores son ampliamente utilizados por entusiastas de Arduino y Raspberry Pi por lo que la documentación respecto a cómo usarlos se encuentra bastante difundida en la Web.

3.6 Subsistema de comunicaciones

Para este subsistema se utilizó conexión a red WiFi ya que opera de forma nativa en las Raspberry Pi Zero W, mientras que en la versión sin WiFi basta con agregar un adaptador USB como se mencionó en el punto 3.4.

La particularidad, es que mediante el lenguaje de programación Python se puede establecer un canal de comunicaciones entre un servidor y un cliente, siendo el servidor, la estación de control que se ejecutará en un computador usando Python, y el cliente será el código que ejecute de forma interna el HAISE-Sat. Con este canal de comunicaciones implementado, se comandará al satélite y se descargaran los datos requeridos para simular una experiencia de control satelital en las actividades relativas a su uso como herramienta de aprendizaje.

3.7 Subsistema de Potencia

El diseño preliminar contemplaba la disponibilidad de componentes en el mercado y los requerimientos energéticos que hay en el resto de los subsistemas para la selección de estos. A grandes rasgos, el diseño preliminar considera las 3 funciones: generar, almacenar y distribuir, sin embargo, la generación de energía es netamente demostrativo dado que la autonomía lograda en el HAISE-Sat con su forma de generar energía no es suficiente para mantener un nivel de energía almacenada constante en el satélite de entrenamiento. El diagrama que muestra la lógica del subsistema de potencia en el diseño preliminar se presenta en la Figura 3-29, este esquema no contiene los sensores de corriente que se señalan en el diseño preliminar (Figura 3-1) con tal de simplificar la visualización de cómo se mueve la energía desde una fuente hasta las líneas de voltaje que alimentan a los subsistemas del HAISE-Sat.

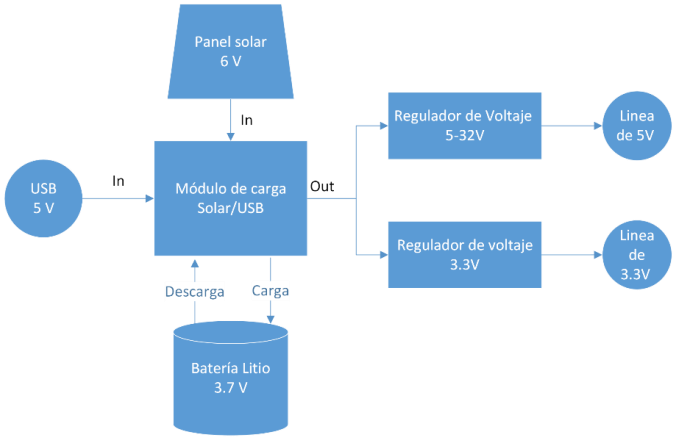


Figura 3-29. Subsistema de potencia del diseño preliminar.

3.7.1 Problemas con el diseño preliminar

Un componente fundamental en este circuito es el Módulo de carga Solar/USB ADA390 visto en la Figura 3-30.a, su función es tomar una tensión de entrada ya sea por USB o por un panel solar, y recargar una batería mientras entrega una tensión de salida donde se puede conectar una carga (por carga se entiende cualquier circuito que requiera de una tensión y una corriente para funcionar), si se retira la fuente de energía, USB o Panel solar, usa la energía almacenada en la batería para alimentar el circuito que esté conectado a la salida del módulo de carga ADA390. Cuando el voltaje de la batería es inferior a 3.1 V, corta el suministro para proteger el estado de la batería (Adafruit, 2022c).

Al realizar las pruebas a cada componente, se encontraron inconvenientes con la combinación del módulo de carga y el regulador de voltaje XL6009.

a) **b)**

Figura 3-30. a) Módulo de carga solar ADA390, b) Regulador de voltaje XL6009

Las pruebas realizadas a los componentes se pueden encontrar en el ANEXO A.1.

En conclusión, el diseño preliminar es insuficiente para integrar el subsistema de potencia y hace falta realizar un nuevo diseño tomando en cuenta los problemas encontrados.

Con todo esto se establecen nuevas consideraciones para el rediseño del subsistema de potencia las cuales son las siguientes.

1. El panel solar será demostrativo y no proveerá energía al subsistema de potencia dada las complicaciones con el rango de voltajes obtenido al usar el ADA390.
2. El regulador XL6009 no será utilizado.
3. El diseño debe asegurar que en presencia de tanto USB como solo batería, entregue los voltajes requeridos en todo momento o mientras la batería lo permita según el estado de carga.

3.7.2 Segunda iteración del subsistema de potencia

Para obtener un nuevo diseño del subsistema de potencia se realizó una búsqueda en el mercado de componentes que cumplan con las consideraciones indicadas en el punto anterior. Para obtener una etapa que regule la tensión de salida a 5 V se optó por circuitos que admitan entradas hasta un máximo de 5 V o un poco más y aseguren en la salida 5 V en todo momento, dos candidatos son los indicados en la Tabla 3-3.

Tabla 3-3 Opciones regulador de voltaje 5 V.

	Componente	
Característica indicada por distribuidor	Adafruit TPS61023 (Adafruit, 2022b)	DC-DC Mini Boost Step Up[3] (Mini Boost)
Tensión de entrada [V]	2 – 5	2.5 – 5.5
Corriente máxima salida [A]	1	1
Tensión de salida [V]	5	5

Las pruebas con el Mini Boost arrojaron buenos resultados, ya que mantuvo la tensión de 5 V en la salida, aun cuando la batería utilizada en las pruebas tenía un nivel de carga bajo (3.4 V batería sola). Este regulador se alimentó además con un puerto USB y se obtuvieron corrientes desde 114 μA (sin nada conectado) hasta 500 mA con carga resistiva sin problemas por lo que cumple con las exigencias del carril de alimentación de 5V indicadas en el PIA (Villarroel, 2022).

También se realizaron pruebas con un módulo TPS61023, Figura 3-31.a, teniendo igual buenos resultados, además de tener un formato en su placa de circuitos más amigable con el diseño eléctrico, por lo que este módulo será utilizado para este subsistema por la falta de documentación en el otro módulo.

Para la etapa de 3.3V se utilizará el componente seleccionado en el PIA (Villarroel, 2022), el cual corresponde al convertidor de voltaje TLV62569 de Adafruit. Las pruebas con este componente concluyen que satisface la función atribuida, la cual es obtener una salida de 3,3V, el rango de corrientes que manejó va desde los 109 μA sin nada conectado hasta 300 mA sin problemas (1.2 A máximo. según fabricante). Este conversor se puede ver en la Figura 3-31.b.

Como se indicó anteriormente y en el PIA (Villarroel, 2022), el subsistema de potencia tendrá sensores de corriente y voltaje, midiendo distintos componentes los cuales corresponden a:

- Línea de 5 V
- Línea de 3.3 V
- Batería
- Panel Solar, que será simulado por un *Light Dependant Resistor*, o LDR

[3] No hay información del fabricante más allá que la entregada por los vendedores.

a) TPS61023 **b) TLV62569**

Figura 3-31. Regulador de 5V y 3.3V (Adafruit, 2022a)

El propósito de esto es que el usuario pueda conocer el estado del subsistema de potencia y lo que es más importante, poder establecer una alerta en el software de vuelo cuando el nivel de carga de la batería sea cercano al voltaje en el que el módulo ADA390 corta el suministro.

El sensor de corriente a utilizar será el INA219, este módulo permite medir de 0 a 26 V y corrientes hasta 3.2 A con una precisión de $\pm 8\,mA$ según distribuidor (DAOKI, 2022). Utiliza el bus de comunicación I2C que está presente en Raspberry Pi por lo que se pueden automatizar las mediciones mediante código.

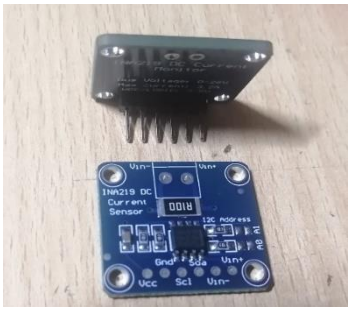


Figura 3-32. Sensor de corriente y voltaje INA219.

La forma de conectar este sensor se indica en el ANEXO A.

3.7.3 Tercera iteración del subsistema de potencia

Para solventar los problemas con el rango de voltajes que había a la salida del módulo de carga ADA390, se adquirió una versión actualizada de este componente, corresponde al módulo bq24074 fabricado por Adafruit, el cual tiene las mismas prestaciones con la salvedad de que el voltaje de salida que tiene está limitado a 4.4V, esto facilita el uso de los reguladores de 5 y 3.3V al limitar la tensión al rango que estos pueden trabajar en sus tensiones de entrada, eliminando los problemas que surgen al tener el panel solar conectado junto a la batería. Así, él panel Solar si bien cumplirá su rol de cargar la batería, su efecto será bastante reducido respecto al consumo eléctrico del HAISE-Sat (una media

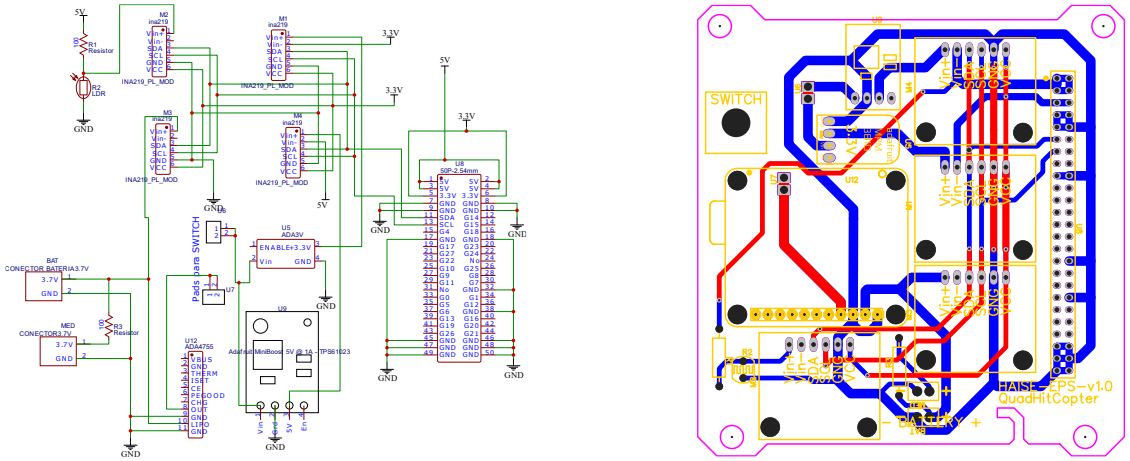
de 2.5W medido), la potencia de la que dispone el panel solar adquirido (menor que 1W) lleva a que sea insuficiente para mantener el nivel de carga de la batería mientras funciona por lo que seguirá siendo demostrativo.

Un inconveniente encontrado en el funcionamiento de tanto el ADA390 como el bq24074 es con el manejo de la batería, ya que si se medía la corriente que circulaba hacia y desde la batería, cuando esta superaba un umbral de aproximadamente 350mA, la batería era desconectada sin razón aparente, probablemente se deba a la resistencia utilizada para medir corriente dentro de los sensores INA219, ya que al ir en serie con la batería, existe una caída de voltaje en dicha resistencia y esa diferencia de voltaje sería suficiente para que los controladores de carga de ambos módulos de carga, interpretaran esto de forma negativa. La solución fue no medir corriente en la batería y solo medir el voltaje al conectar un solo terminal del sensor a la batería, sin interrumpir el circuito.

Algunos aspectos adicionales al diseño son los siguientes:

- Dada la forma en que está integrado el conector del panel solar, dificultaba poder medir las variables eléctricas de este componente, así que en esta primera versión del HAISE-Sat se cambió el punto de medición por un *Light Dependant Resistor* o LDR, el cual genera el mismo efecto que medir un panel solar cuando cambia la intensidad lumínica que recibe.
- Para efectos de su uso como herramienta de aprendizaje se incorporó un interruptor al diseño que desconecta al resto de subsistemas del suministro eléctrico, esto, por ejemplo, para poder cargar la batería sin necesidad de encender la Raspberry Pi y resto de componentes.
- Se incorporó un punto de medición de la batería, esto para actividades de verificación que se realicen al ser usado como herramienta de aprendizaje.
- El contorno de la placa incluye el espacio para el panel solar incorporado en el diseño del subsistema estructural.

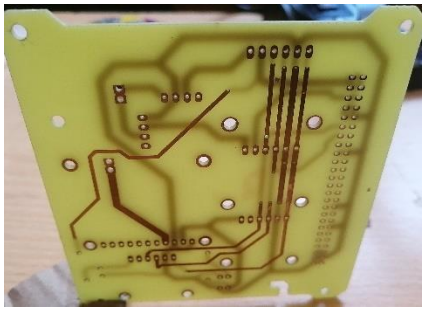
Finalmente, el diseño eléctrico del subsistema de potencia se puede observar en la Figura 3-33, el proceso de diseño fue realizado en el software gratuito EasyEDA (*EasyEDA Webpage*, 2022).



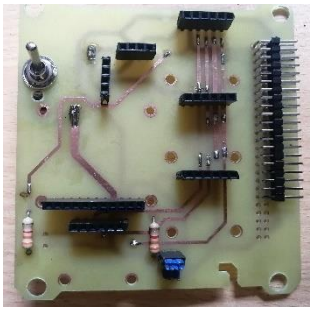
a) Diagrama eléctrico **b) Diseño eléctrico PCB**

Figura 3-33. Diseño eléctrico del subsistema de potencia.

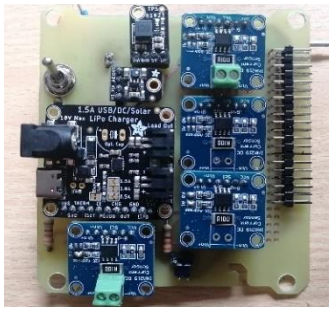
Una vez fabricada la placa en el C4i, se puede observar su apariencia salida del C4i, y con los componentes integrados al PCB en la Figura 3-34.



a) Producto salido del C4i



b) Placa con conectores y componentes fijos



c) EPS con módulos integrados

Figura 3-34. Primera versión de la placa de circuitos del subsistema de potencia fabricada.

3.7.4 Batería del subsistema de potencia.

En el PIA (Villarroel, 2022), el diseño preliminar propone el uso de una batería de litio de 3.7 V, cuya capacidad sea la suficiente para dar 1 hora de autonomía. Con el módulo de carga USB bq24074 se pueden cargar baterías de polímero de litio (Li-Po) y de iones de litio (Li-Ion). Para la integración del HAISE-Sat se utilizaron baterías Li-Ion en el empaquetado estandarizado 18650, dado su fácil instalación y retiro para almacenaje del satélite de entrenamiento cuando no se esté utilizando, estas baterías tienen el aspecto indicado en la Figura 3-33 y dado su empaquetado, facilita la recarga de estas fuera del satélite con cargadores especializados en este tipo de baterías.

Figura 3-35. Batería Li-Ion de referencia (Papis et al., 2020).

De igual forma que con el modelo 3D del ArduSat, existen modelos para porta baterías de este estándar para su impresión 3D, uno en particular se obtuvo de la página ThingiVerse , la misma de donde se obtuvo los archivos del ArduSat, y su aspecto se aprecia en la Figura 3-36, esto irá adherido en la parte posterior (Figura 3-34.b) de la placa del subsistema de potencia.

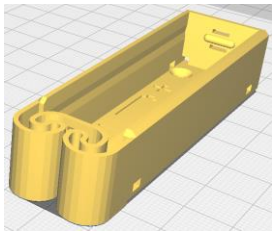


Figura 3-36. Porta baterías 18650, recuperado de (Thingiverse, 2022).

Un aspecto importante de la batería es su capacidad energética medida en mAh, indica cuanta energía es capaz de almacenar. Para el HAISE-Sat se estimó un consumo de corriente de 600 mA en la línea de 5 V según el PIA (Villarroel, 2022), lo que usando relaciones eléctricas deja un consumo de aproximadamente 800 mA para la batería a 3.7 V como voltaje promedio durante el uso del satélite de entrenamiento. Se debe verificar que la batería sea capaz de entregar esta corriente con un margen aceptable, esto dependerá del fabricante de la batería y la máxima corriente de descarga que permita la batería ya que es necesario asegurar el consumo continuo de corriente de cada uno de los subsistemas con tal de que se mantenga encendido en cualquier estado de requerimiento energético por parte del HAISE-Sat, en el mercado se observan ejemplares que admiten hasta 7 A como corriente de descarga.

Dado que el HAISE-Sat está diseñado para ser utilizado en un ambiente académico, es razonable establecer que la autonomía del satélite sea cercana a las 2 horas, tomando como consumo medio 800 mA en la batería, se puede estimar la capacidad de la batería de la siguiente forma:

$$C_{bat} = (I_{bat} \cdot t)/\epsilon \ [mAh] \quad \textbf{(1)}$$

Con t el tiempo requerido en horas, I_{bat} la corriente que se requiere de forma constante en mA y C_{bat} la capacidad de la batería, usando los valores indicadores anteriormente se estima una capacidad de 1600 mAh, que si se aplica un factor de eficiencia debido a las pérdidas de energía durante la transformación y transporte, tomando un valor bastante conservador, ϵ, del 70% , la capacidad requerida sube a aproximadamente 2300 mAh como mínimo, con tal de funcionar sin interrupciones durante el transcurso de las actividades desarrolladas en su uso como herramienta de aprendizaje.

3.7.5 Porta baterías para el subsistema de potencia.

Una vez que se verificó que el modelo visto en la Figura 3-36 permitía instalar una batería 18650 y proveer energía, se generó un diseño a partir de este que permitiera integrarlo a las guías usadas por las placas de circuito de los subsistemas. Así, usando la huella generada en el punto 3.2.3 se ubicaron los agujeros de los espárragos y rigidizando a criterio la estructura el resultado se ve en Figura 3-37.

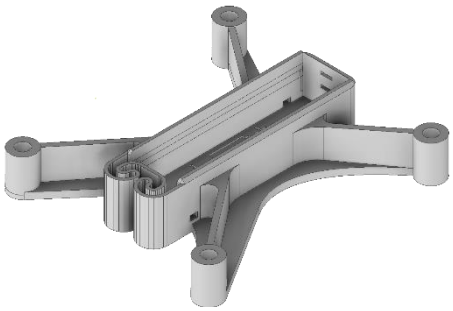


Figura 3-37. Porta batería compatible con la estructura.

3.7.6 Integración y pruebas del subsistema de Potencia.

Con el porta batería y la placa de circuito fabricadas, se integran los componentes del subsistema de potencia, lo cual se puede ver en la Figura 3-38. Posterior a la integración se verifica que las salidas en el conector de este subsistema correspondan a los niveles de tensión de diseño, los resultados de las pruebas se resumen la Se midió además el voltaje en la batería cuando es cargada, arrojando un voltaje de 4.22V de media, el cual es un valor seguro para la carga de esta batería. Como conclusión, el subsistema de potencia cumple con los requisitos eléctricos del satélite de entrenamiento.

Tabla 3-4. Obviando GND, el resto de los pines al no tener nada conectado se verifica que marquen 0V en cada uno de ellos.

Figura 3-38. Integración del subsistema de potencia

Se midió además el voltaje en la batería cuando es cargada, arrojando un voltaje de 4.22V de media, el cual es un valor seguro para la carga de esta batería. Como conclusión, el subsistema de potencia cumple con los requisitos eléctricos del satélite de entrenamiento.

Tabla 3-4. Resultados de mediciones en las pruebas del subsistema de potencia.

Pin	**Valor teórico**	**Valor medido**	
		Solo batería	**Cargando con USB**
5V	5V	5.01V	5.02V
3.3V	3.3V	3.32V	3.33V
SDA	3.3V	3.30V	3.31V
SCL	3.3V	3.30V	3.31V

Cuando el interruptor, desde ahora en adelante *Kill Switch*, corta el suministro, la línea de 5V se reduce a 0.2V mientras que el resto baja a 0V, esto se puede deber a la presencia de un condensador en el regulador de 5V que mantiene una pequeña cantidad de energía almacenada en él, pero no supone un problema ya que se terminará de descargar al estar conectados el resto de los subsistemas, este voltaje no es suficiente para que se haga un falso encendido del HAISE-Sat.

3.8 Prototipado del HAISE-Sat en Protoboard.

En paralelo al trabajo realizado en el diseño de los subsistemas, se fue probando el diseño eléctrico usando una Protoboard donde se conectaban todos los componentes mediante cables, esto con el fin de verificar el funcionamiento de los esquemas eléctricos generados. El diseño eléctrico final en Protoboard se puede ver en la Figura 3-39, posterior a esta verificación, se empezó la fabricación de las placas de circuito y segunda iteración del Subsistema Estructural.

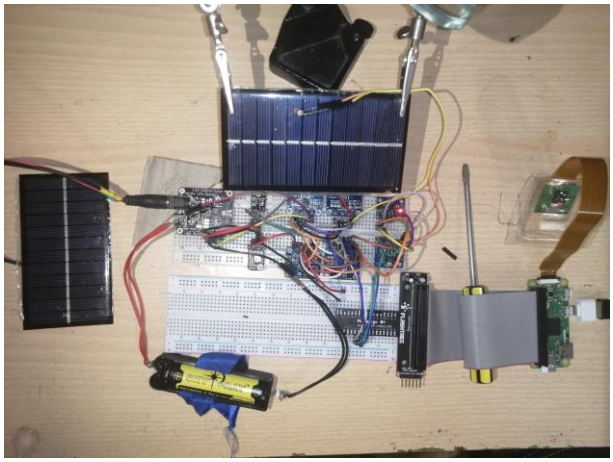


Figura 3-39. Prototipado del diseño eléctrico en Protoboard.

3.9 Diseño del *Backplane*, placa de conexiones.

Para poder conectar las dos placas con subsistemas, es necesario diseñar y fabricar la placa que servirá de interfaz entre ellas, lo que en los satélites del programa BIRDS se denomina *Backplane Board* o BPB, para ello se obtuvieron las dimensiones que debiesen tener las placas de los subsistemas, con tal de determinar la distancia real entre los conectores de 50 pines, así, dando un margen de 3 mm respecto a la altura del subsistema de potencia y 1 mm respecto a la altura a la placa con el C&DH, se sitúan en una placa de circuitos 3 conectores, uno para el subsistema de potencia, otro para la placa de circuitos que contiene al C&DH, y un conector extra que permitirá integrar otros sensores en revisiones futuras del HAISE-Sat. Estos 3 conectores tienen la misma asignación de pines, sin embargo, poseen una diferencia importante respecto al conector que llevan y es que son la reflexión en espejo de los conectores de los subsistemas, para que permita realizar las conexiones de forma perpendicular y facilitar la fase de diseño con el software EasyEDA.

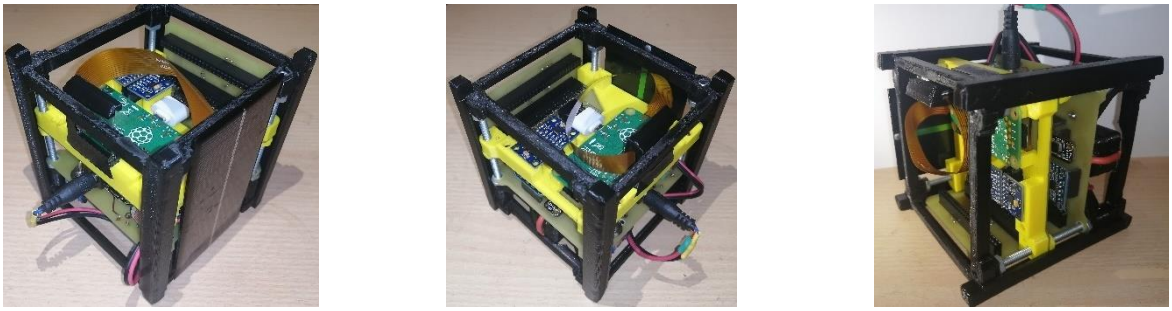
La forma de esta placa está determinada por el espacio del que dispone dentro de la estructura, y dado que en la ubicación que se posee dentro del CubeSat Bus queda muy cercana a las bases, la dimensión de esta placa de circuitos coincide con el volumen libre que dejan las bases.

Se adicionaron puntos de medición como complemento para la realización de actividades con esta herramienta de aprendizaje. La placa de circuitos diseñada en EasyEDA, fabricada e integrada se ven en la Figura 7-5 del ANEXO A.

3.10 Integración del HAISE-Sat.

Con todos los subsistemas integrados y el Backplane fabricado, se procede a integrar el satélite, los pasos para la integración se detallan en el ANEXO B. Así, la primera versión del HAISE-Sat completamente integrado a nivel de Hardware se puede ver en la Figura 3-40. En la tercera imagen,

c), se puede apreciar la lógica de integración presente en los satélites del programa BIRDS (Kim et al., 2021).



a) **b)** **c)**

Figura 3-40. HAISE-Sat integrado.

3.10.1 Verificación de requisitos.

La verificación de los requisitos y restricciones se lista en la Tabla 3-5.

Tabla 3-5. Verificación de Requisitos y Restricciones.

ID	Método	Acción tomada	Estado
C.1	Inspección	Las dimensiones medidas en el HAISE-Sat se mantiene con un margen de error del 2% respecto a las establecidas por el Estándar y la masa corresponde al 20% de la masa máxima establecida por el estándar, ver Anexo F para más detalles.	Verificado
C.3	Inspección	Cada pieza fue fabricada dentro de la Facultad de Ingeniería al igual que la integración y pruebas.	Verificado
C.4	Análisis	Las modificaciones respetan las restricciones de diseño del trabajo previo (Villarroel, 2022).	Verificado
R.1	Demostración	Se utilizó el satélite en dos salas de la Facultad, exhibiendo sus capacidades de forma completa.	Verificado
R.2	Demostración	Al integrar el satélite, se evidencia el *Backplane* como interfaz eléctrica similar a la utilizada en UNISEC (Klaus Schilling & Stephan Busch, 2017).	Verificado
R.3	Inspección	El conector adicional está en el diseño eléctrico y permite conectar sensores, sin embargo, no se probó con una placa de circuitos por lo que no se verifica en su totalidad el requisito.	Falta evidencia
R.4	Demostración	Se midió el estado de la batería en dos modos de operación hasta que el satélite se apaga, uno por falta de energía y otro con un apagado programado midiendo el estado de la batería, resultados en Anexo F.	Verificado

CAPITULO 4: Propuesta de experiencia de aprendizaje

Con el satélite de entrenamiento construido, faltan dos elementos importantes para que este sea de utilidad, una guía de uso y Software necesario, en este capítulo se definirán características relevantes para las actividades donde se puede utilizar al HAISE-Sat como material de apoyo para el estudiante. Las actividades serán basadas en trabajos como el HEPTA-Sat (Yamazaki & Zengo, 2018) y EYAS-Sat (EyasSat, 2016) entre los más destacables.

4.1 Estructura de la experiencia

Una experiencia de aprendizaje orientada a satélites debe permitir al estudiante construir una base de conocimientos sobre los conceptos utilizados en Ingeniería Satelital y espacial, además de un dominio técnico en la arquitectura de satélites al llevar a cabo tareas de integración y verificación de requisitos en *Hardware*, debe permitir entender al satélite como una parte de un sistema, reconociendo los factores involucrados en el funcionamiento de estos.

Así, se pueden destacar dos etapas: la primera orientada a entregar los conocimientos relativos al concepto y funcionamiento de los satélites, destacando los aspectos positivos y negativos de los satélites tipo CubeSat al compararlo con satélites más grandes; y la segunda, enfocada en proveer al estudiante con una experiencia *Hands-on Learning*, la cual consiste en aprender haciendo, interactuando con material de estudio que represente de buena forma el tópico de interés, en el caso de esta experiencia, utilizar al HAISE-Sat como material de apoyo.

Del plan de trabajo expuesto por Yamazaki para el programa de entrenamiento basado en el HEPTA-Sat (Yamazaki & Zengo, 2018), se recupera un diagrama en la figura Figura 4-1. Aquí la primera etapa se evidencia en el paso *Understand Functions*, mientras que *Assemble*, *Integration* y *Test* engloban la segunda etapa.

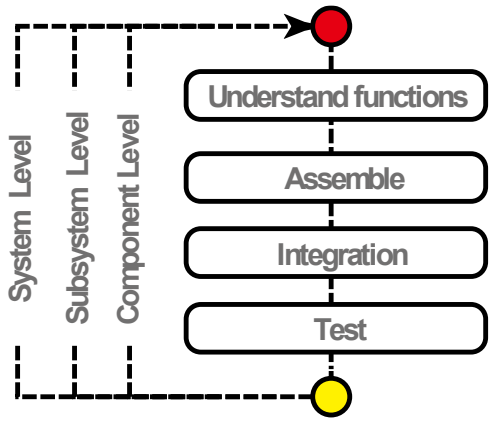


Figura 4-1. Flujo de trabajo utilizado en el programa HEPTA-Sat (Yamazaki & Zengo, 2018).

De la figura se entiende que, en el programa HEPTA-Sat comienzan con la entrega de conocimientos sobre satélites, partiendo con la descomposición funcional de un satélite y como se alocan estas funciones a subsistemas, componentes y partes con una asignación física de estos en la herramienta de aprendizaje HEPTA-Sat, así, a medida que se van integrando cada subsistema, se va realizando un análisis a nivel de subsistema, entendiendo la función que cumplen los componentes que componen a cada subsistema, luego se integra y hacen pruebas con el satélite completamente integrado para realizar un análisis a nivel de sistema, lo cual contempla todo lo necesario para que el satélite pueda cumplir su función, esto involucra el hardware requerido para operar al satélite y el software tanto en la estación de control como el software a bordo del satélite.

4.2 Primera etapa: Entrega de conocimientos

Al comenzar la experiencia de aprendizaje *Hands-on learning* con el HAISE-Sat, se entregarán conocimientos sobre la arquitectura funcional y física de satélites, tanto convencionales como el caso particular de los CubeSat, que es la lógica bajo la cual se sustenta el HAISE-Sat. Adicional a la arquitectura de satélites, el alumno deberá ser capaz de comprender el contexto en el cual se desarrollan los *Small Satellites*, incluyendo la posición de estos en las distintas nomenclaturas utilizadas para referirse a satélites artificiales y cuáles son las razones que justifican adoptar una nomenclatura por sobre otra, incluyendo la posición del estándar CubeSat.

Para esta primera etapa, puede ser necesario tener conocimientos sobre mecánica orbital, con tal de permitir al estudiante relacionar algunas características del satélite y la misión que cumple, con el ambiente espacial en el que se encuentra orbitando, esto con el fin de lograr una mejor integración de los conocimientos recibidos.

Esta etapa, puede llevarse a cabo con clases presenciales, en línea, o asincrónicas, con la preparación de un material y la guía del docente a cargo de la experiencia de aprendizaje.

El orden en que se puede implementar la entrega de conocimientos sobre los subsistemas puede ser mediante la secuencia Clase-Laboratorio, donde se entregan los fundamentos del subsistema en cuestión y le sigue inmediatamente un laboratorio donde manipula dicho subsistema, realizando la integración y pruebas de este.

4.3 Segunda etapa: Experiencia *Hands-on*

En esta etapa, el estudiante manipulará el hardware del HAISE-Sat, deberá realizar un chequeo de cada componente del HAISE-Sat y verificar que funciona a nivel de componente. Posteriormente el estudiante procederá a integrar los subsistemas del satélite y realizar una verificación de requisitos para cada subsistema, en el caso del HAISE-Sat, los subsistemas de CD&H, Payload y comunicaciones van en una sola placa por lo que, a nivel de subsistema, la verificación se hará en la placa de circuitos que los contiene.

Usando como guía el programa HEPTA-Sat y las actividades descritas en la guía de usuario del EYAS-Sat, una propuesta de actividades para la experiencia *Hands-on learning* se presenta en la Tabla 4-1.

Tabla 4-1. Guía de actividades para el HAISE-Sat, propuesta inicial.

Paso 1: Verificación de componentes	Verificar que estén todos los componentes requeridos para la integración y operación del HAISE-Sat.	
Paso 2: Integración de subsistema de potencia (EPS)	Analizar la función del EPS y diagrama eléctrico	
		Verificar estado de las baterías (medir)
		Verificar estado del panel solar (medir)
		Verificar módulo de carga de baterías (medir)
		Masar cada componente del EPS y registrarlo
	Integrar los componentes del EPS en su respectiva placa de circuitos y verificar el funcionamiento midiendo donde corresponda.	
		Verificar línea de 5V
		Verificar línea de 3.3V
		Verificar 0V en los pines que corresponde
		Verificar funcionamiento del *Kill Switch*
		Verificar voltaje de batería durante carga
		Verificar voltaje de panel solar
Paso 3: Integración EPS con Backplane	Integrar el EPS a la placa de distribución o Backplane verificando cada pin del conector midiéndolo.	
		Verificar línea de 5V
		Verificar línea de 3.3V
		Verificar 0V en los pines que corresponde
		Verificar línea I2C
Paso 4: Integración de subsistemas: C&DH, payload y comunicaciones	Pruebas de la Raspberry Pi	
		Preparar software en computadora
		Cargar software a la Raspberry Pi
		Verificar conectividad WiFi
		Conectar Cámara y verificar funcionamiento
	Integrar Raspberry y sensores a la placa de circuitos	

Paso 5: Pruebas de subsistemas sin estructura.	Integrar EPS y placa de subsistemas (C&DH, comunicaciones, payload) en el Backplane	
		Verificar voltaje en los pines (medir)
		Verificar comunicación entre estación de control (computador) y satélite (HAISE-Sat).
		Apagar y desensamblar placas de circuito
Paso 6: Integración y pruebas de subsistemas con estructura	Seguir la secuencia de integración (Anexo B)	
		Encender satélite integrado y verificar comunicación con estación de control.
		Verificar telemetría de potencia y comparar con mediciones.
		Verificar canal de comunicaciones de payload (Captar y enviar imagen), mediante telecomando.
		Cambiar de posición u orientación al satélite y repetir la verificación de telemetría y captación de imágenes.

Estos pasos considerarían un uso básico del kit del HAISE-Sat al ser aplicado a una experiencia de aprendizaje. Si lo que se busca es una experiencia más específica como control satelital, se puede modificar el código que controla al satélite y estación de control con tal de simular fallos de componentes, los cuales tengan que ser verificados y para solucionarlos se deba enviar un telecomando.

Un telecomando consiste en una instrucción denotada por una secuencia de caracteres generalmente alfanuméricos, que se envían al satélite desde la estación de control y el satélite, en su programación tendrá una acción a seguir al recibir dicho telecomando.

Además, es posible modificar el código con tal de que se puedan cargar planes de trabajo al satélite y automatizar la captura de imágenes y almacenamiento de datos, por ejemplo para un margen de tiempo, almacenar los datos de telemetría y cada 3 minutos capture una imagen, para al finalizar los el tiempo programado, descargue los datos de telemetría y payload a la estación de control, inclusive simulando las ventanas de visibilidad que tendría el satélite a determinada altura orbital, estas ventanas de visibilidad corresponden a periodos en donde la antena es capaz de comunicarse con el satélite.

CAPÍTULO 5: Software del HAISE-Sat

Para poder operar este satélite de entrenamiento, es necesario que cuente con un software cargado, y a su vez, que un computador haga de estación de control estando ambos conectados a una red que posea WiFi. En este capítulo se detallará la arquitectura del software implementado para volver operativo al HAISE-Sat.

Todo el software fue desarrollado en Python, lo que lo vuelve un proyecto Open-Source sumado a que el hardware fue diseñado e integrado con componentes COTS con la salvedad de las piezas fabricadas por Impresión 3D.

Dos librerías de Python, importantes en el código del satélite y estación de control son las librerías *threading* y *socket*, la primera es utilizada para poder ejecutar múltiples funciones a la vez de forma simultánea, lo que permite que se realicen varias tareas operando de manera independiente las unas de las otras; la segunda librería es la responsable de permitir establecer un canal de transmisión de datos entre dos dispositivos que están conectados a una misma red, así, al conectar a la red WiFi el satélite, es posible comunicarse con él de manera inalámbrica facilitando la manipulación del satélite de entrenamiento.

5.1 Software de vuelo a bordo del HAISE-Sat

La estructura del código a bordo del satélite se corresponde con la Figura 5-1, esquemáticamente se deja a la lectura de sensores como un solo bloque, sin embargo en el código todo es programado mediante funciones concretas, es decir, existe una función que lee sensores de voltajes, una función que lee los sensores de movimiento inercial (IMU) y otra función que se ejecuta cuando el telecomando para tomar una imagen es recibido por el satélite, de igual forma para los canales de comunicación.

Cada lectura de un sensor involucra guardar los datos obtenidos en la memoria, en el caso de los sensores de voltaje e IMU, son datos que son guardados desde que se inicia el satélite en un mismo archivo, en esta versión del software este archivo siempre se sobrescribe cada vez que se inicia el satélite, sin embargo, para versiones futuras es posible programarlo de tal forma que mantenga un número predeterminado de archivos de registro

Los canales de comunicación son unidireccionales, es decir solo envían datos en un solo sentido, esta solución fue utilizada para simplificar la realización de tareas periódicas que no requieren de entradas del usuario, como lo es la visualización de la telemetría en la estación de control, es decir que mientras el satélite esté comunicado con la estación, estará enviando datos de la telemetría sin necesidad de esperar una respuesta de la estación, no así la captura de imágenes, tarea que se ejecutará solo al recibir el telecomando designado, así mismo, el canal de descarga de datos del payload y *Housekeeping* (registro del estado del satélite) solo se abre cuando hay una solicitud de captura de imagen o datos

del satélite y posteriormente es cerrado. De esta forma, los únicos canales que permanecen activos en todo momento mientras el HAISE-Sat esté encendido son el canal de telecomandos y de telemetría.

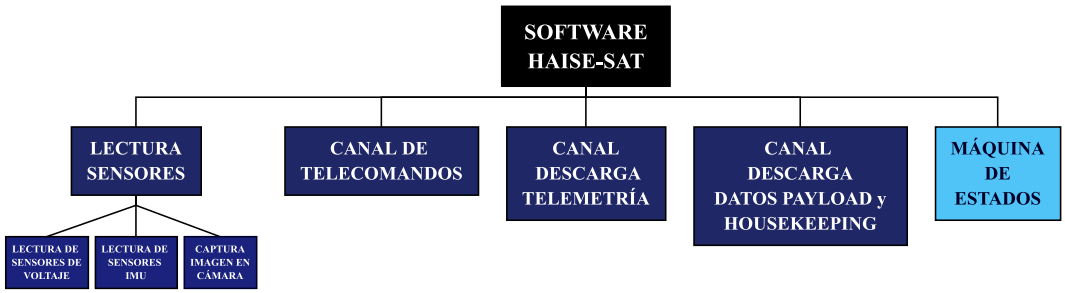


Figura 5-1. Estructuración del programa que ejecuta HAISE-Sat.

El último segmento indicado en la Figura 5-1 es responsable de registrar que está haciendo el satélite, en la versión actual solo posee dos estados, NORMAL y CAPTURE, el primero representa al satélite enviando telemetría de forma continua, con todos sus sensores operativos, el segundo indica que el satélite está tomando una fotografía mientras todo el resto de los subsistemas está funcionando de manera correcta. Bajo esta lógica, es posible modificar el programa para que simule fallos de componentes y que se requiera de una acción para remediarlo, como puede ser enviar una instrucción al satélite para indicarle que el fallo encontrado no supone mayores problemas para la misión o implementar protocolos más sofisticados para manejo de errores similares a los utilizados en satélites convencionales.

Se deben realizar algunas modificaciones cada vez que se utilice el satélite en una red WiFi diferente, esto se debe a que la función encargada de comunicar la estación de control con el satélite requiere conocer la dirección IP del computador donde se ejecuta la estación, además de necesitar las credenciales para acceder a dicha red WiFi, esta tarea se realiza al modificar un archivo y está detallada en el ANEXO B. El código que ejecuta el satélite se encuentra en el ANEXO C.

5.2 Software para implementar la estación de control

El código Python que se ejecuta en la estación de control se puede describir en la Figura 5-2, posee una parte dedicada a la interfaz de usuario, la cual corresponde a la forma en que el usuario podrá ingresar telecomandos y visualizar el estado del satélite, además de ver la imagen que el HAISE-Sat haya capturado cuando se envía el telecomando correspondiente, además posee líneas de código dedicadas a mantener un canal de comunicaciones abierto para la descarga de la telemetría y una función que abre un canal de comunicaciones para descarga de datos más complejos como lo son imágenes y archivos de registro. El código implementado está en el ANEXO D.

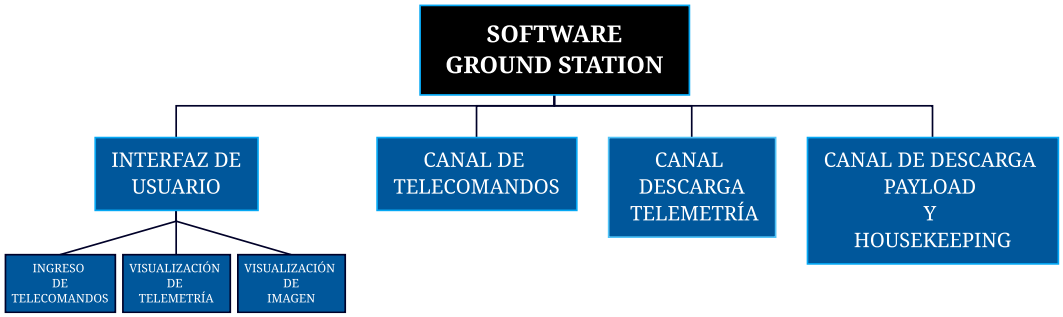


Figura 5-2. Estructura del programa que ejecuta la estación de control.

El funcionamiento del software de la estación de control es, una vez ejecutado el código, inicia un servidor el cual esperará hasta que el satélite se conecte, esto sucede segundos después de encender al satélite. Con la conexión lograda, se abre la interfaz de usuario, donde se podrán ingresar telecomandos en forma de texto, y se podrá visualizar el estado del subsistema de potencia, y lo que están midiendo los sensores IMU. Al ingresar el telecomando lo que sucede dentro del programa es que se registra en un archivo el telecomando y la fecha en la que se envía, posteriormente una función verifica que el texto ingresado corresponda a un telecomando existente dentro del programa y lo envía por el canal logrado con el satélite, una vez que el satélite lo recibe, enviará de vuelta el último telecomando recibido, con tal de reconocer que el canal está funcionando.

Para el telecomando de captura de imagen o descarga de datos *Housekeeping*, el satélite recibirá dicha instrucción y enviará mediante un canal dedicado a la descarga de datos del *Payload* y *Housekeeping* el archivo correspondiente, la estación de control lo recibirá y lo mostrará en pantalla. Tanto las imágenes como el archivo de *Housekeeping* quedarán almacenados en la ubicación del archivo Python que ejecuta la estación de control.

CONCLUSIONES

Este trabajo concluye la primera iteración del proyecto iniciado con el Proyecto de Ingeniería Aeroespacial, el cual tenía por objetivo diseñar y construir un producto que pudiera ser implementado en cursos de ingeniería satelital o de *Systems Engineering*, partiendo desde la identificación de necesidades de un *Stakeholder* para definir los requisitos que debiese tener este producto, como resultado se obtuvo el diseño preliminar de un CubeSat Bus, el cual simula las características de un CubeSat real, dada las facilidades y ventajas que han tenido estos al momento de implementarlos dentro de carreras universitarias por sus bajos tiempos de desarrollo a diferencia de satélites convencionales que podrían tardar años antes de dar luz verde al lanzamiento y posterior operación. Además, al aprovechar estos reducidos tiempos de desarrollo, permiten que nueva tecnología sea puesta a prueba en el espacio, así mismo con realización de experimentos para investigación científica, permitiendo que por ejemplo estudiantes de pregrado y grado puedan ganar experiencia en misiones espaciales, ya sea en el desarrollo del satélite o en las distintas misiones que se puedan llevar a cabo (Marzioli et al., 2020).

Respecto a los objetivos del proyecto, al finalizar este trabajo se concluye que:

1. El diseño obtenido al iterar sobre el trabajo previo resultó con una verificación satisfactoria, las funciones correspondían a las que exigía el árbol de funciones FBS, estas eran:
   - El subsistema de potencia, al ser capaz de suministrar la energía requerida por el prototipo.
   - El subsistema de comandos y manejo de datos al ser capaz de ejecutar comandos directos en la Raspberry Pi.
   - En conjunto con el C&DH, se verificó el funcionamiento de los sensores correspondientes a la cámara y acelerómetros, permitiendo además leer los sensores de voltaje y corriente del subsistema de potencia, dejando registro de ello en la memoria de la Raspberry Pi.
2. Al verificarse la factibilidad del diseño, la construcción del HAISE-Sat fue el siguiente paso, así al construir cada uno de los subsistemas y verificando el cumplimiento de sus funciones asignadas, la integración del satélite de entrenamiento resultó en una buena primera iteración de esta herramienta de aprendizaje, que, al incorporar el Software, se pudo verificar el funcionamiento de este "Sistema Satelital" simulado, al contar con una estación terrestre que pudiese controlar al satélite mediante la captura de imágenes con la cámara, descarga de telemetría e imágenes, pudiendo visualizarse estas en la estación terrestre. Estas operaciones se ejecutan en un computador el cual hace de estación terrestre o *Ground Station*, pudiendo correr el Software en computadores con Windows que admitan Python 3.0 y tengan conexión a una red de área local o LAN con WiFi.
3. Se diseña una experiencia de aprendizaje en base a las ya existentes en la bibliografía, identificando puntos relevantes para la formación en ingeniería de satélites, donde cada actividad permite que el estudiante interactúe con el Hardware del HAISE-Sat
4. Si bien se verificó el funcionamiento del satélite dentro de aulas de la Facultad de Ingeniería de forma satisfactoria, hizo falta utilizarlo en la experiencia de aprendizaje diseñada con tal de recibir una retroalimentación de los alumnos involucrados.

Por parte de la verificación de restricciones y requisitos se destaca:

- Se verificó la restricción C.1, la cual corresponde a que el satélite este restringido al estándar CubeSat en volumen, masa y forma (Cal Poly, 2020). El volumen se mantiene dentro de un margen de error del 2% respecto a las dimensiones indicadas por el estándar, error que es aceptable dentro de las capacidades de la impresión 3D y no suponen un mayor inconveniente para la función del HAISE-Sat como herramienta de aprendizaje, mientras que la forma es acorde a lo recomendado por el estándar. La masa cumple las restricciones dadas por el estándar y deja un buen margen para implementar mejoras al diseño.
- La restricción C.3 y C.4 fueron cumplidas, el satélite fue construido con los procesos de fabricación disponibles en la Universidad, destacando al C4i (Centro para la Industria 4.0) por la fabricación de las placas de circuito, y cada modificación realizada al diseño preliminar, sigue cumpliendo las restricciones iniciales de diseño del Proyecto de Ingeniería Aeroespacial (Villarroel, 2022).
- Para los requisitos, el R.1 se verificó satisfactoriamente al operar el satélite en distintas aulas de la Universidad, en el Laboratorio de Técnicas Aeroespaciales y en el edificio Tecnológico Mecánico se pudo verificar el funcionamiento del satélite, esto se logró conectando al HAISE-Sat a las redes WiFi y enviando telecomandos desde un computador portátil, recibiendo los datos que enviaba el satélite de forma correcta.
- El requisito R.2 se logra al incorporar al diseño la placa de circuitos denominada *Backplane*, el cual emula de buena manera la integración vista en los CubeSat desarrollados bajo la propuesta de estándar eléctrico de UNISEC, por otro lado el requisito R.3 se cumple parcialmente al incorporar el conector adicional para futuros subsistemas y verificando las señales eléctricas en cada pin del conector, pero al no contar un subsistema adicional para integrarlo no se puede determinar el correcto funcionamiento de este y el cumplimiento total del requisito, sin embargo diversos sensores fueron probados en los pines correspondientes y funcionaron de forma correcta.
- Otro de los requisitos de diseño heredados del PIA, corresponde al requisito R.4, el cual es la autonomía del HAISE-Sat, esta se verificó de forma satisfactoria con el satélite integrado en dos modos de operación, una donde el satélite consume la batería hasta apagarse, y un segundo modo donde se programó un voltaje seguro de operación de la batería para que apague la Raspberry Pi (la cual contempla el mayor consumo energético de todos los componentes integrados), logrando más de 7 horas de autonomía en ambos modos, cumpliendo así con las 2 horas establecidas como requisito.

Durante la fase de integración se identificaron algunos puntos débiles del diseño, los cuales pueden ser resueltos con nuevas iteraciones tales como el volumen que ocupa el Subsistema de Potencia o la falta de rigidez de la estructura al momento de cumplir su función de mantener todo en su lugar. El subsistema de potencia requiere una revisión enfocada en optimizar el espacio, por ejemplo, utilizando un porta baterías que vaya soldado a la placa y reubicar los sensores con tal de hacer a este subsistema más compacto y simplificando su integración, al realizar este cambio se gana el suficiente espacio para integrar una placa adicional a este satélite de entrenamiento, el cual puede ser conectado a la interfaz eléctrica adicional que tiene el Backplane, este espacio podría ser aprovechado con un sistema de control de actitud para realizar futuros estudios sobre dinámica en satélites.

Por otro lado, en el Software son evidentes muchas falencias y oportunidades para hacer de esta herramienta de aprendizaje un elemento valioso en la formación de profesionales para la tan necesitada industria espacial en el país, madurar la programación de este satélite de entrenamiento recibiendo retroalimentación de usuarios resulta una oportunidad de mejora, ya que no requiere una reinversión para reemplazar hardware.

Así, en el desarrollo de este trabajo se ha iterado sobre el diseño preliminar con tal de poder fabricar la primera versión del HAISE-Sat, se integraron conocimientos de electrónica, programación, procesos de fabricación e ingeniería satelital para conseguir un producto que tenga valor en el proceso de aprendizaje de la ingeniería satelital y espacial. He de destacar que, dada las distintas disciplinas involucradas en el desarrollo de satélites, futuras iteraciones pueden tener un mejor desempeño si se ataca el problema desde cada una de estas disciplinas como lo son la Electrónica, la Mecánica, las Telecomunicaciones y la programación, con un equipo que tenga un alto dominio de cada una de estas ramas, de esta forma se podrá pulir tanto al producto, el HAISE-Sat, como la experiencia de aprendizaje que se pueda desarrollar con este, proyectando a futuro de que sirva como base para fomentar y facilitar la realización de futuras misiones espaciales diseñadas y operadas por centros de estudio en el país.

Referencias

Aboaf, A. P., Harrod, E. S., Zola, M., Prakash, A., Palo, S. E., Marshall, R., Pilinski, M. D., Rainville, N., Dahir, A., Nataraja, V., Schwab, B., Gardell, A. & Warshaw, L. (2020). *A Methodology for Successful University Graduate CubeSat Programs*.

Adafruit. (2022a). *Adafruit 3.3V Buck Converter Breakout TLV62569*. https://www.adafruit.com/product/4711

Adafruit. (2022b). *Adafruit MiniBoost 5V @ 1A - TPS61023*. https://www.adafruit.com/product/4654

Adafruit. (2022c). *Módulo de carga solar/USB Adafruit*. https://www.adafruit.com/product/390

Ashish MIshra. (2021). *Basic GPIO Control on Raspberry Pi Zero W*. https://circuitdigest.com/microcontroller-projects/basic-gpio-control-on-raspberry-pi-zero-w-blinking-an-led

Azami, M. H., Maeda, G., Faure, P., Yamauchi, T., Kim, S., Masui, H. & Cho, M. (2019). BIRDS-2: A Constellation of Joint Global Multi-Nation 1U CubeSats. *Journal of Physics: Conference Series*, *1152*(1). https://doi.org/10.1088/1742-6596/1152/1/012008

Bockel, J.-M. (2018). *The Future Of The Space Industry*. https://www.nato-pa.int/document/2018-future-space-industry-bockel-report-173-esc-18-e-fin

Botelho A. S., R. C. & Xavier, A. L. (2019). A Unified Satellite Taxonomy Proposal Based on Mass and Size. *Advances in Aerospace Science and Technology*, *04*(04), 57–73. https://doi.org/10.4236/aast.2019.44005

Bouwmeester, J., Langer, M. & Gill, E. (2017). Survey on the implementation and reliability of CubeSat electrical bus interfaces. *CEAS Space Journal*, *9*(2), 163–173. https://doi.org/10.1007/s12567-016-0138-0

Bryce Space and Technology. (2022). *Smallsats by the Numbers 2022*. https://brycetech.com/reports

Busch, S. & Schilling, K. (2016). *Standardization Approaches for Efficient Electrical Interfaces of CubeSats*. https://www.researchgate.net/publication/354837502_Standardization_Approaches_for_Efficient_Electrical_Interfaces_of_CubeSats

Cal Poly. (2020). *CubeSat Design Specification Rev. 14 The CubeSat Program, Cal Poly SLO CubeSat Design Specification*. https://www.cubesat.org/cds-announcement

CubeSatShop. (2022a). *CubeSatShop: webshop for CubeSats and Nanosats*. https://www.cubesatshop.com/

CubeSatShop. (2022b). *ISIS ISIPOD 3-Unit CubeSat deployer*. https://www.cubesatshop.com/product/3-unit-cubesat-deployer/

DAOKI. (2022). *Sensor de corriente INA219 I2C*. https://www.amazon.com/-/es/INA219-alimentaci%C3%B3n-corriente-bidireccional-MCU-219/dp/B07X956BBC

EasyEDA Webpage. (2022). https://easyeda.com/es

ECSS Secretariat. (2012). *ECSS Glossary of terms*. http://everyspec.com/ESA/ECSS-S-ST-00-01C_47906/

European Space Agency. (2022). *Get your own satellite in orbit in just 10 months*. https://www.esa.int/Applications/Technology_Transfer/Get_your_own_satellite_in_orbit_in_just_10_months

EyasSat. (2016). *GEN 5 Nanosatellite Simulator, with COSMOS, User Guide*. http://eyassat.com/resources-downloads/gen-5-nanosatellite-simulator-user-guide/#_Toc459719155

Filho, A. C. J., Tikami, A., Paula, E. de S. F. de, Piñeros, J. M., Fernandes, G. F., Camargo, L. A. P., Santos, C. A. M. B. dos, Santos, W. A. dos & Naccarato, K. P. (2020). CubeSat Development for Lightning Flashes Detection: RaioSat Project. *Journal of Aerospace Technology and Management*, *12*(12), 80–93. https://doi.org/10.5028/jatm.cab.1161

Goodwin, J. S. & Wegner, P. (2001). Evolved expendable launch vehicle secondary payload adapter - helping technology get to space -. *AIAA Space 2001 Conference and Exposition*. https://doi.org/10.2514/6.2001-4701

Gunter's Space Page. (2022a). *OneWeb Satellite information*. https://space.skyrocket.de/doc_sdat/oneweb.htm

Gunter's Space Page. (2022b). *Radarsat-2 satellite information*. https://space.skyrocket.de/doc_sdat/radarsat-2.htm

Gunter's Space Page. (2022c). *Starlink satellite information*. https://space.skyrocket.de/doc_sdat/starlink-v1-0.htm#:~:text=The%20Starlink%20satellites%20feature%20a,use%20krypton%2Dfueled%20Hall%20thrusters.

INCOSE. (2020). *Guide to the Systems Engineering Body of Knowledge (SEBoK), version 2.2*. www.sebokwiki.orgPDFgeneratedusingtheopensourcemwlibtoolkit.Seehttp://code.pediapress.com/formoreinformation.

ISISPACE. (2022). *CubeSatShop: ISIS 1-Unit CubeSat structure*. https://www.cubesatshop.com/product/1-unit-cubesat-structure/

Iwai, H. (2021). (Gordon E. Moore Award) Impact of Micro-/Nano-Electronics, Miniaturization Limit, and Technology Development for the Next 10 Years and After. *ECS Transactions*, *102*(4), 81–95. https://doi.org/10.1149/10204.0081ecst

Jerry Sellers. (2011). *EyasSAT: A Classroom Nanosatellite for Teaching Space Systems Engineering*. http://unisec.jp/nanosat_symposium/1st/files/10th.PM/Session_4/Presentation_Jerry-Sellers.pdf

Kim, S., Cho, M., Masui, H. & Yamauchi, T. (2021). BIRDS BUS: A Standard CubeSat BUS for an Annual Educational Satellite Project. In *JoSS* (Vol. 10, Issue 2). www.jossonline.com

Kiselyov, I. (2020). *Model-Based Reliability Engineering*. https://repository.tudelft.nl/islandora/search/kiselyov?collection=education

Klaus Schilling & Stephan Busch. (2017). *CubeSat Subsystem Interface Definition CSID (Proposal)*. http://unisec-europe.eu/wordpress/wp-content/uploads/CubeSat-Subsystem-Interface-Standard-V2.0.pdf

Kramer, A. & Schilling, K. (2021). First demonstration of collision avoidance and orbit control for pico-satellites — UWE-4. *Acta Astronautica*, *185*, 244–256. https://doi.org/10.1016/j.actaastro.2021.04.010

Kumar, K., Pauline, F., Maeda, G., Kim, S., Masui, H. & Cho, M. (2018). *BIRDS-2: Multi-Nation CubeSat Constellation Project for Learning and Capacity Building*. https://digitalcommons.usu.edu/smallsat/2018/all2018/435/

Lay, K. S., Li, L. & Okutsu, M. (2022). High altitude balloon testing of Arduino and environmental sensors for CubeSat prototype. *HardwareX*, *12*, e00329. https://doi.org/10.1016/J.OHX.2022.E00329

Madry, S., Martinez, P. & Laufer, R. (2018). Innovative Design, Manufacturing and Testing of Small Satellites. In *Innovative Design, Manufacturing and Testing of Small Satellites* (pp. 105–111). Springer International Publishing. https://doi.org/10.1007/978-3-319-75094-1_8

MAKERFAM. (2012). *ArduSat - The Arduino CubeSat Satellite (full scale model)*. https://www.thingiverse.com/thing:27300

Marzioli, P., Frezza, L., Amadio, D., Hossein, S. H., Pancalli, M. G., Picci, N., Vestito, E., Piergentili, F., Celesti, P., Curianò, F., Gugliermetti, L. & Santoni, F. (2020). Hands-on education through nano-satellites development: past, current and future projects at Sapienza S5Lab; Hands-on education through nano-satellites development: past, current and future projects at Sapienza S5Lab. In *2020 IEEE 7th International Workshop on Metrology for AeroSpace (MetroAeroSpace)*.

Maskey, A., Lepcha, P., Shrestha, H. R., Chamika, W. D., Malmadayalage, T. L. D., Kishimoto, M., Kakimoto, Y., Sasaki, Y., Tumenjargal, T., Maeda, G., Kim, S., Masui, H., Yamauchi, T. & Cho, M. (2022). One Year On-Orbit Results of Improved Bus, LoRa Demonstration and Novel

Backplane Mission of a 1U CubeSat Constellation. *Transactions of the Japan Society for Aeronautical and Space Sciences, 65*(5), 213–220. https://doi.org/10.2322/tjsass.65.213

MinCiencia. (2022). *Programa Nacional Espacial*.

NASA. (2017). *CubeSat 101: Basic Concepts and Processes for First-Time CubeSat Developers*.

NASA. (2018). *ELaNa XIX CubeSat Launch on Rocket Lab Electron Mission*.

NASA. (2020). *Small Spacecraft Technology State of the Art*. https://www.nasa.gov/sites/default/files/atoms/files/soa2020_final3.pdf

ORBCOMM. (2016). *Constelación de satélites de última generación OG2 de ORBCOMM*. https://www.orbcomm.com/es/networks/satellite/orbcomm-og2

Papis, K., Figaj, R., Kuś, J., Żołądek, M. & Zając, M. (2020). Application of photovoltaic cells as a source of energy in unmanned aerial vehicle (UAV) – case study. *E3S Web of Conferences, 173,* 02002. https://doi.org/10.1051/e3sconf/202017302002

Pilcher, J. L. (2021). *Comparison of Decision Analysis Methods for a CubeSat Propulsion System*. https://scholar.afit.edu/etd/4954

Poghosyan, A. & Golkar, A. (2017). CubeSat evolution: Analyzing CubeSat capabilities for conducting science missions. *Progress in Aerospace Sciences*, *88*, 59–83. https://doi.org/10.1016/j.paerosci.2016.11.002

Raspberry Pi Foundation. (2018a). *Raspberry Pi Documentation*. https://www.raspberrypi.com/documentation/computers/

Raspberry Pi Foundation. (2018b). *Raspberry Pi Zero Pinout* . https://aprendiendoarduino.wordpress.com/category/curso-raspberry-pi/

Rocket Lab. (2022). *Rocket Lab: Mission ELaNa-19 of NASA*. https://www.rocketlabusa.com/missions/completed-missions/elana-19/

Ryschkewitsch, M., Schaible, D. & Larson, W. (2009). The Art and Science of Systems Engineering. In *Systems Research Forum* (Vol. 3, Issue 2). www.worldscientific.com

Thingiverse. (2022). *Li-Ion 18650 Battery Holder*. https://www.thingiverse.com/thing:456900

Torres, R. (2017). *Satélite de observación de la Tierra FASat-Charlie*. https://obtienearchivo.bcn.cl/obtienearchivo?id=repositorio/10221/24637/1/Sat%C3%A9lite_de_observaci%C3%B3n_de_la_Tierra_FASat-Charlie.pdf

Villarroel, B. P. (2022). *Desarrollo de un diseño preliminar de CubeSat Bus para entrenamiento académico*.

Woellert, K., Ehrenfreund, P., Ricco, A. J. & Hertzfeld, H. (2011). Cubesats: Cost-effective science and technology platforms for emerging and developing nations. *Advances in Space Research*, *47*(4), 663–684. https://doi.org/10.1016/j.asr.2010.10.009

Yamazaki, M. (2015). Hands-on learning of space systems engineering by using classroom pico-satellite HEPTA. *2015 7th International Conference on Recent Advances in Space Technologies (RAST)*, 825–829. https://doi.org/10.1109/RAST.2015.7208454

Yamazaki, M. (2018a). *HEPTA-Sat Training Program: International Knowledge Transfer Using Hands-on Type CubeSat Education*.

Yamazaki, M. (2018b). *HEPTA-Sat Training Program: International Knowledge Transfer Using Hands-on Type CubeSat Education*.

Yamazaki, M. & Zengo, T. (2018). *HEPTA-Sat Training Program: International Knowledge Transfer Using Hands-on Type CubeSat Education*.

ANEXO

A Diseño de subsistemas

A.1 Pruebas del diseño preliminar del subsistema de potencia.

El primer inconveniente observado al analizar el diseño preliminar está en el regulador de voltaje que debe entregar 5 V en todo momento. El regulador de voltaje usado se ve en Figura 3-30.b, corresponde a un circuito basado en el chip XL6009, su modo de operación es tomar un voltaje de entrada y regularlo a una tensión deseada a la salida, esto mediante el ajuste de una resistencia variable dentro de su circuito. El problema surge cuando la tensión de entrada es inferior a 3.5V, si la tensión de entrada baja de este valor, la tensión en la salida adopta valores cercanos a 10V de forma no constante lo cual es suficiente para dañar los componentes del resto de subsistemas.

El comportamiento real del ADA390, depende de si está conectado vía USB, por panel solar o si solo está usando la energía almacenada en la batería como se ve en la Tabla 7-1. Esto no fue considerado en el diseño preliminar, y lleva a que la salida del regulador de voltaje XL6009 se tengan voltajes distintos según la fuente de energía que se esté utilizando. Esto es de suma importancia, ya que todos los componentes del resto de los subsistemas son susceptibles a la tensión de alimentación que tengan, pudiendo dañarse si el voltaje es muy elevado.

Tabla 7-1 Comportamiento de ADA390.

	Fuente de energía		
	Solo batería (según estado de carga)	Cable USB + batería (Según voltaje del puerto USB)	Panel Solar + batería
Voltaje de salida en ADA390	3.1 – 4.0 V	Igual al voltaje USB: 4.8- 5 V	El mayor voltaje entre el entregado por el panel solar y la batería 3.1-6.5 V [4]

Considerando el comportamiento del módulo ADA390 y el regulador XL6009, el resultado final es que cuando solo está la batería como fuente de energía, la tensión a la salida del ADA390 y por consecuencia, la entrada del XL6009 puede alcanzar un voltaje inferior al admisible, resultando en

[4] Tomando en consideración el mínimo voltaje que la batería de litio puede alcanzar con tal de no deteriorarse, controlado por el módulo ADA390 (Adafruit, 2022c) y la tensión máxima que debe proveer el panel solar en este módulo de carga.

una salida que no cumple con la función del EPS de acondicionar energía, ya que el voltaje requerido es de 5 V el cual es impuesto por los subsistemas.

El segundo inconveniente es que, si se incorpora el panel solar, la entrada al XL-6009 será el indicado en la Tabla 7-1, entre 3.1 y 6.5 V, repitiendo el problema de tener solo batería en el mínimo y teniendo un voltaje no conveniente en el caso del máximo como se verá a continuación.

Para la verificación del regulador de voltaje se utilizaron dos XL-6009 como se observa en la Figura 7-1. El regulador A será utilizado para verificar el comportamiento del XL-6009 con diferentes entradas. Inicialmente ambos circuitos están ajustados para que, teniendo una entrada de 4.92 V, la que entrega el puerto USB al cual está conectado, la tensión de salida sean aproximadamente 5 V.

La primera medición es al voltaje de entrada del regulador A en la Figura 7-1.

Luego se realizan cuatro mediciones que se pueden ver en la Figura 7-2. En la Figura 7-2.d se simula la presencia del panel solar en el circuito al ajustar a 6.5 V la entrada del regulador B, esto demuestra que la tensión de salida del XL6009 depende de la tensión que tiene como entrada lo cual no es una característica deseada dado el comportamiento del módulo de carga ADA390 que varía la tensión de salida en función de la fuente de energía.

También se verifica el funcionamiento cuando la tensión de entrada es inferior a 3.5 V con una batería descargada la cual se observa en la Figura 7-3. Considerando que el consumo de corriente del XL6009 sin una carga aplicada es bastante reducido, en los ejemplares utilizados para estas pruebas se midió la corriente y no supera los 6mA, se descarta que la potencia disponible en la batería sea la limitante del problema.

En conclusión, el diseño preliminar es insuficiente para integrar el subsistema de potencia y hace falta realizar un nuevo diseño tomando en cuenta los problemas encontrados.

Con todo esto se establecen nuevas consideraciones para el rediseño del subsistema de potencia las cuales son las siguientes.

1) El panel solar será demostrativo y no proveerá energía al subsistema de potencia dada las complicaciones con el rango de voltajes obtenido al usar el ADA390.
2) El regulador XL6009 no será utilizado.
3) El diseño debe asegurar que en presencia de tanto USB como solo batería, entregue los voltajes requeridos en todo momento o mientras la batería lo permita según el estado de carga.

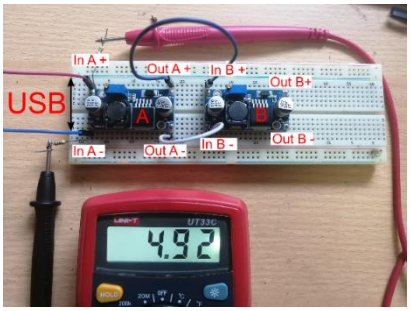
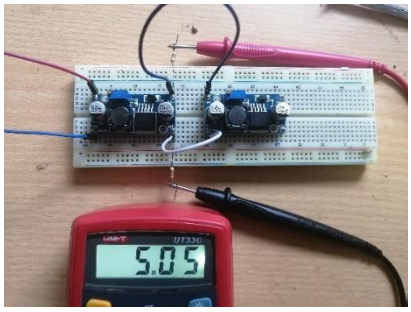
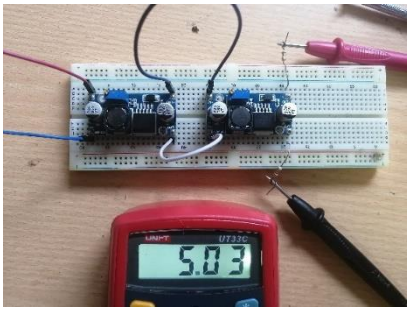


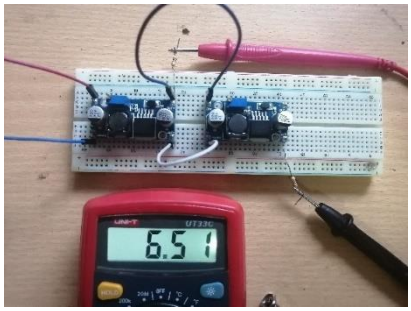
Figura 7-1. Configuración para pruebas del XL6009



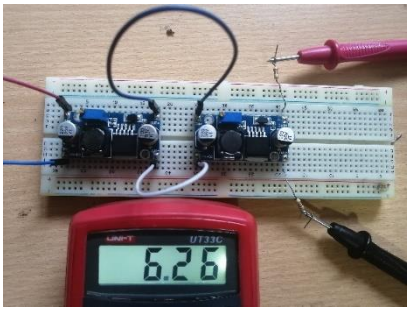
a) Salida de A regulado a ~ 5 V



b) Salida de B con A regulado a ~ 5 V

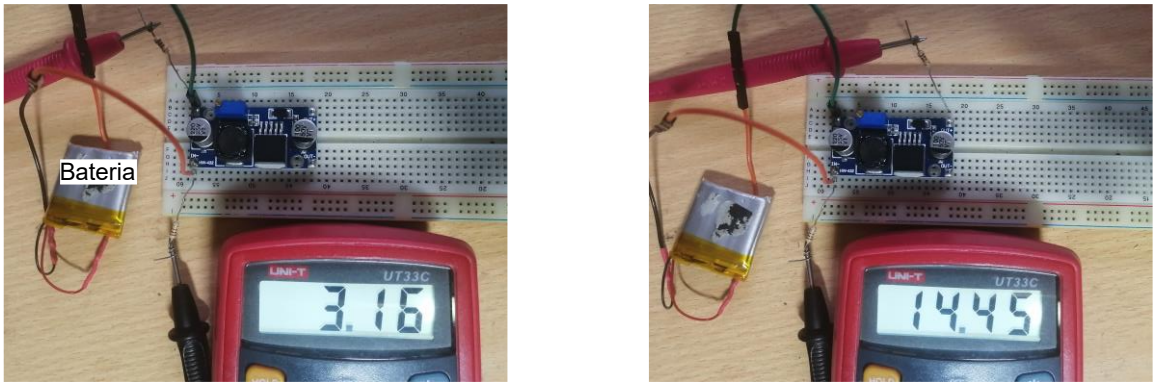


c) Salida de A regulada a ~ 6.5 V



d) Salida de B con A regulado a ~ 6.5 V

Figura 7-2. Mediciones para distintos voltajes de entrada en el XL6009



a) Voltaje en la batería

b) Voltaje a la salida del XL-6009 ajustado para que entregue 5V con 4.92V de entrada

Figura 7-3. Verificación del XL-6009 con tensión inferior al indicado por el fabricante.

A.2 Utilización del sensor de corriente y voltaje INA219

Este sensor se conecta como indica la Figura 7-4: VCC indica la fuente de alimentación que puede ir de 3.3V hasta 5V según el distribuidor (DAOKI, 2022); GND indica la tierra del circuito, en este caso, el polo negativo de la fuente de alimentación; las etiquetas SDA_RPZ y SCL_RPZ indican los pines de la Raspberry Pi a la que se conecta este sensor; R1 representa una carga cualquiera que requiera medirse la corriente. Este sensor mide el voltaje que hay entre el pin $Vin+$ y GND, al mismo tiempo mide la corriente que circula entre los pines $Vin+$ y $Vin-$. Los pines Vin de ambos lados son equivalentes, por lo que puede usarse en ambos. La señal con los datos medidos será enviada por las líneas I2C hasta la Raspberry Pi.

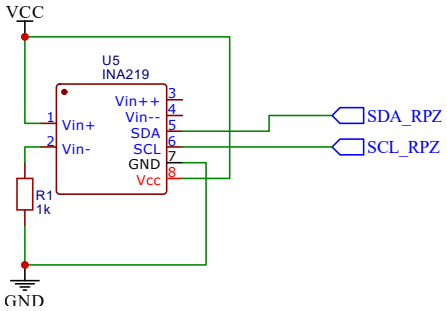


Figura 7-4. Diagrama de conexión del sensor INA219

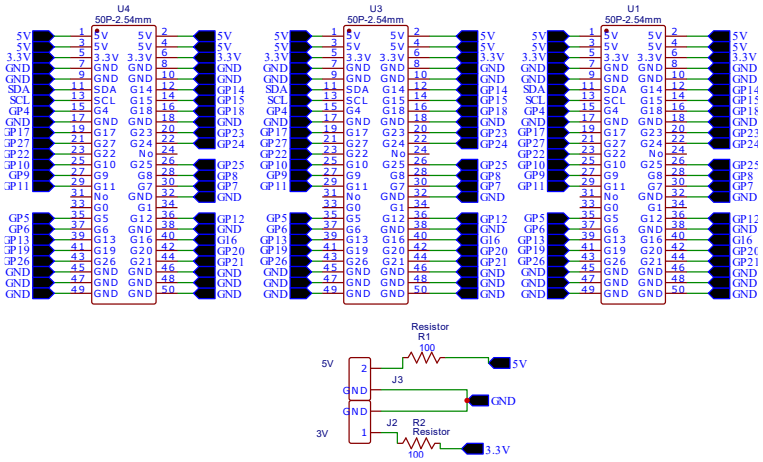
Como se indicó anteriormente, se medirá la corriente y voltaje de 4 puntos, por lo que se requiere 4 sensores INA219 para realizar esta tarea y como se indicó en el punto 3.3, cada dispositivo tendrá un identificador propio dentro del protocolo I2C, para que cada sensor tenga un identificador distinto se deben soldar las conexiones indicadas como A0 y A1 en la placa del INA219, el identificador obtenido

obedece la regla indicada por la Tabla 7-2. Así desde código en Raspberry Pi, se puede acceder a las mediciones generadas por cada sensor de forma separada.

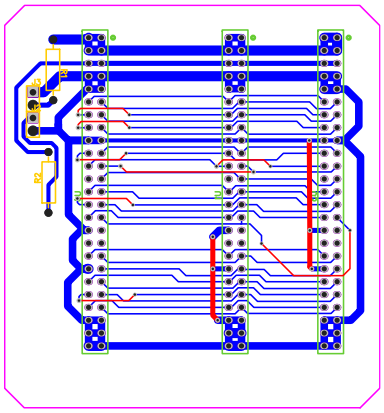
Tabla 7-2. Distinción de identificador para sensor INA219 en protocolo I2C

A0	A1	Identificador
Sin soldar	Sin soldar	0x40
Soldado	Sin soldar	0x41
Sin soldar	Soldado	0x44
Soldado	Soldado	0x45

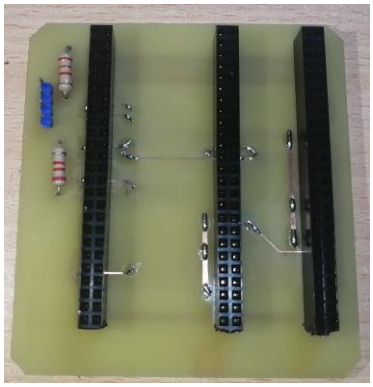
A.3 Diseño de Backplane Board



a) **Diagrama eléctrico**



b) **Placa de circuito impreso**



c) ***Backplane* fabricado**

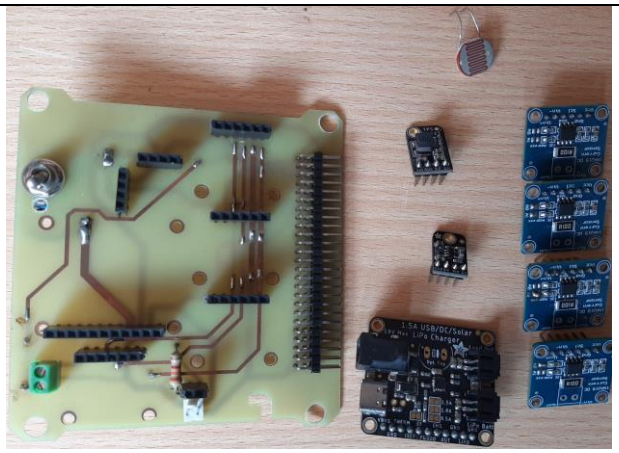
Figura 7-5. Diseño y Fabricación de la placa *Backplane*

B Pasos para la integración del Kit del HAISE-Sat

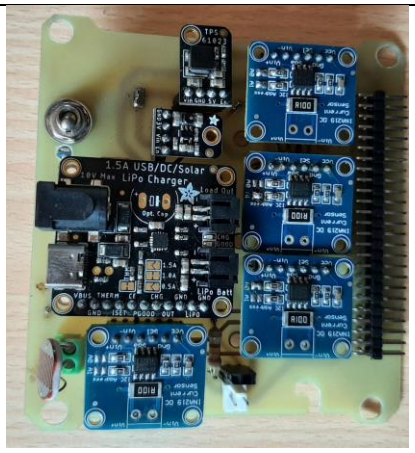
En la Figura 7-6 se observa la composición del kit del HAISE-Sat.

Figura 7-6. Componentes del Kit HAISE-Sat

Con las piezas preparadas se seguirá la siguiente secuencia para integrar el Hardware del HAISE-Sat

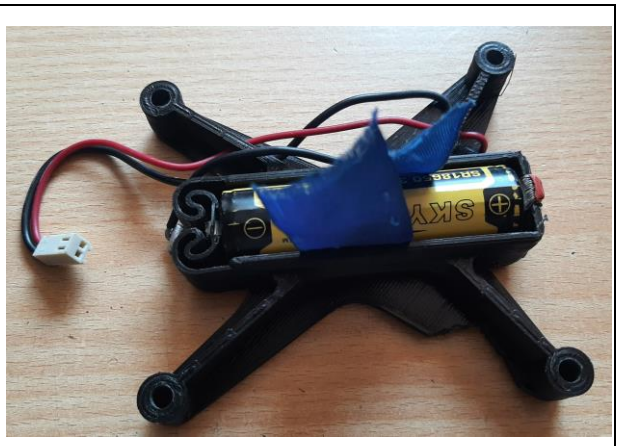


1)

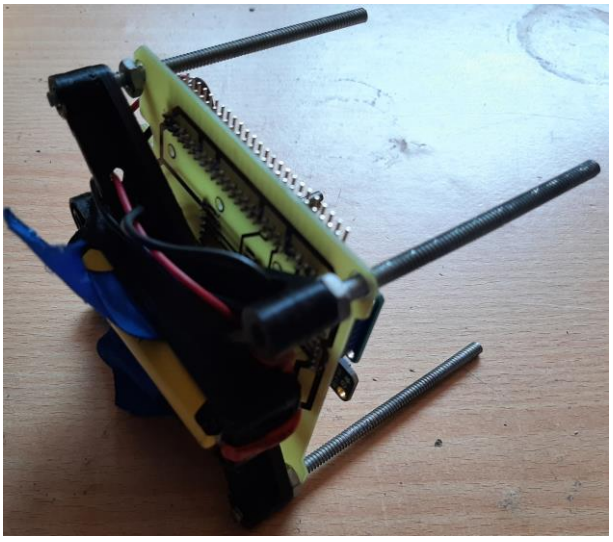


2)

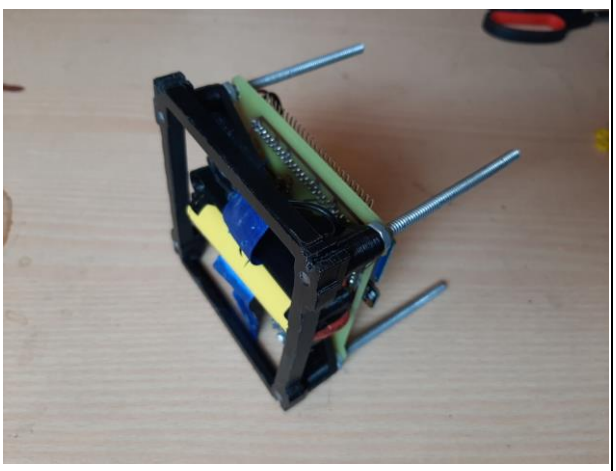
3)



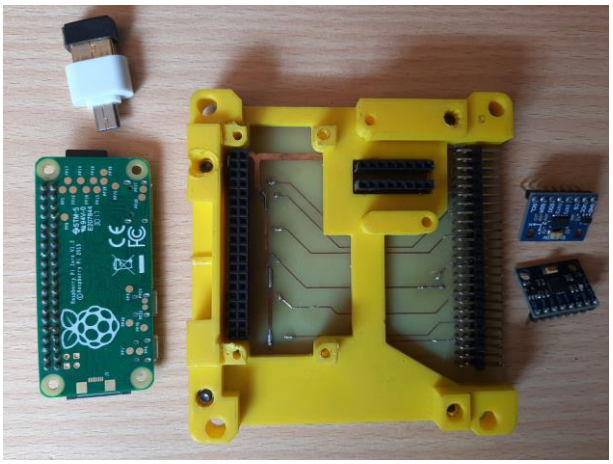
4)



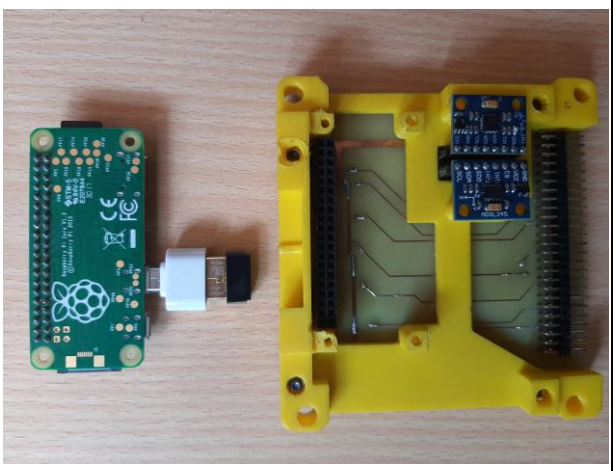
5)



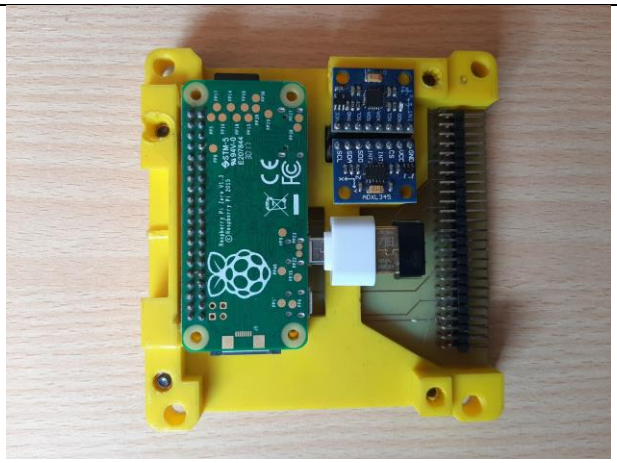
6)



7)



8)

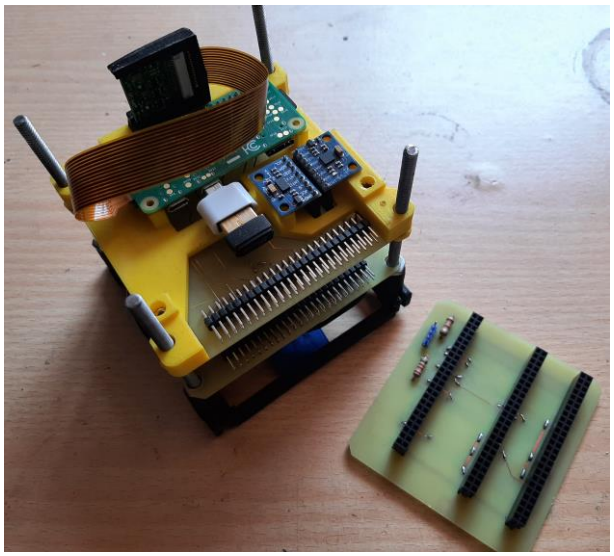


9)

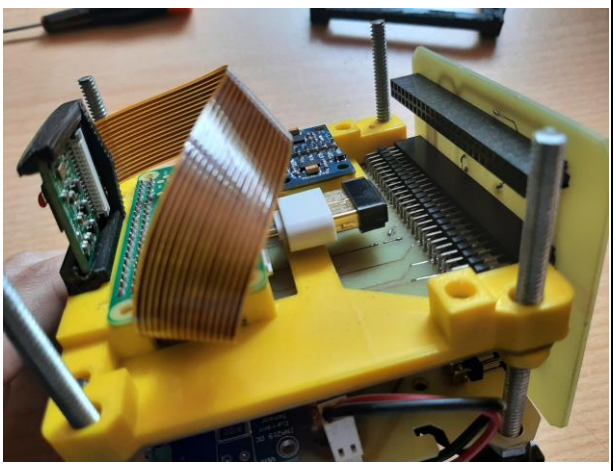
10)

11)

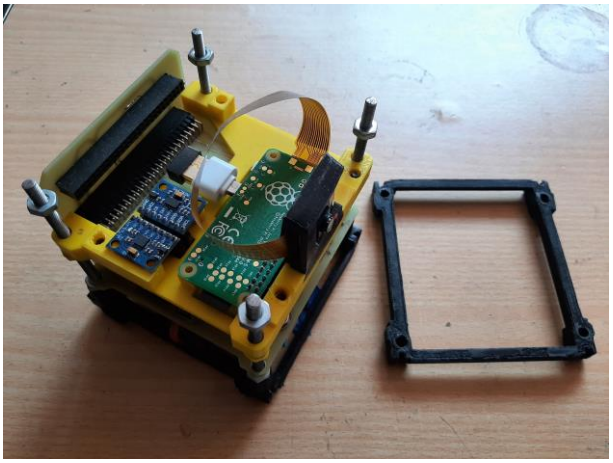
12)



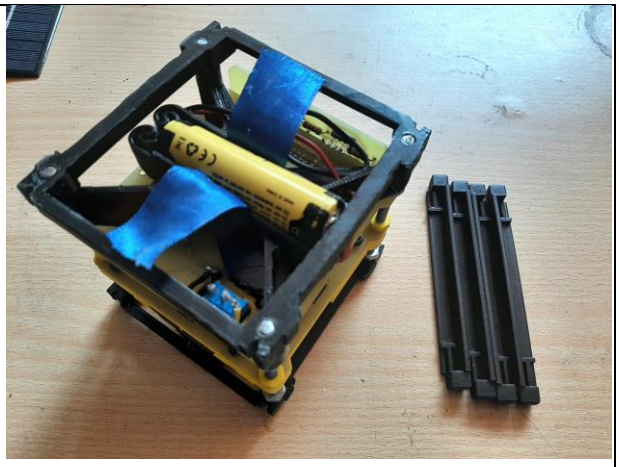
13)



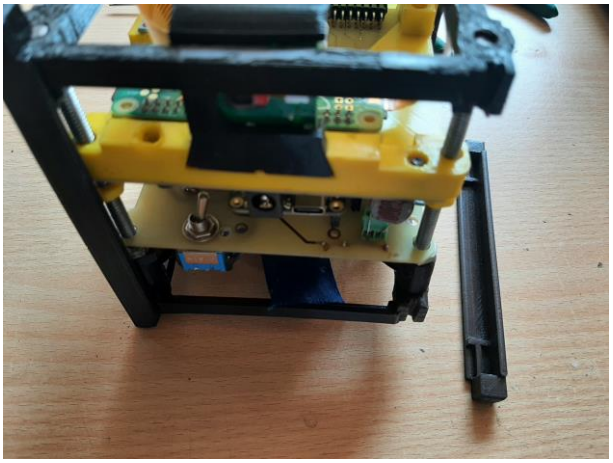
14)



15)

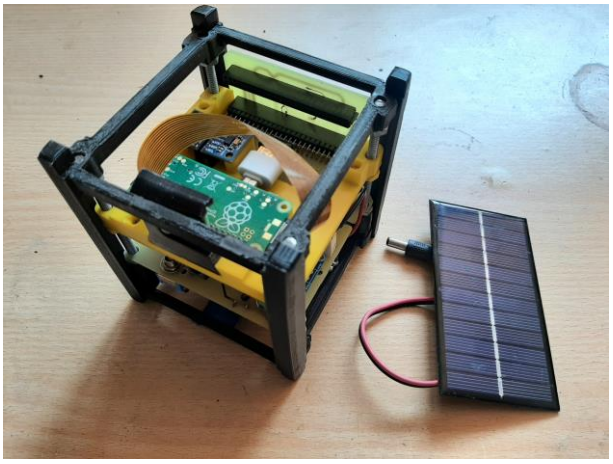


16)

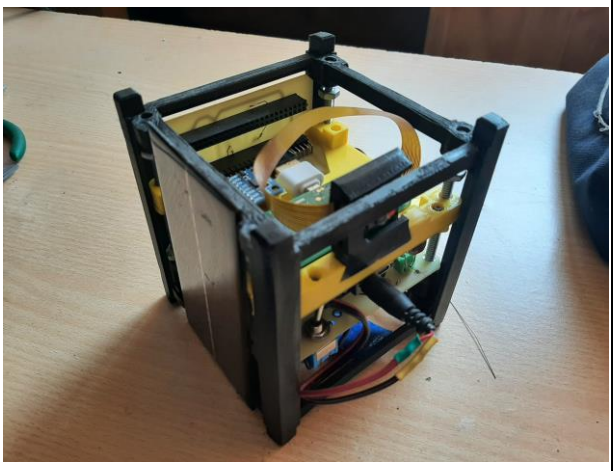


17)

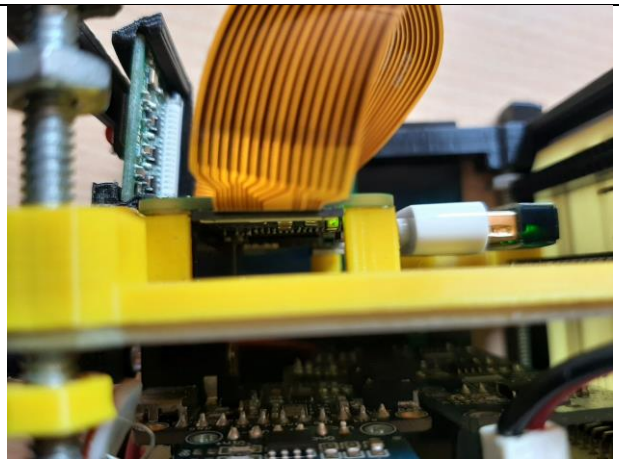
18)



19)



20)



21)

Figura 7-7. Secuencia de integración del Hardware del HAISE-Sat

Pasos:

1) Se inicia integrando el subsistema de potencia, primero se identifican los componentes.
2) Luego se procede a integrar las fases de regulación y adaptación de voltajes, además de los sensores de voltaje y corriente para conocer el estado de la energía.
3) Se identifican los componentes indicados.
4) Se incorpora la batería a su soporte.
5) El soporte de batería y la placa del EPS son fijadas mediante los espárragos y se incorporan tuercas para fijar una distancia entre las dos piezas.
6) Una de las bases es fijada al atornillar los espárragos en los agujeros, notar que la cara plana de la base va hacia afuera.
7) Le sigue integrar la placa que contiene al C&DH, comunicaciones y Payload. Se identifican los componentes requeridos.
8) Se conectan los sensores, el sensor superior corresponde al MPU-6050/GY-52, el inferior es el ADXL345, además se conecta el adaptador WiFi al micro USB de la Raspberry Pi en caso de usar una Raspberry Pi Zero y la memoria microSD.
9) Se integra la Raspberry a la placa.
10) Se identifican las piezas para integrar la cámara.
11) Se fija el soporte de la cámara a la placa en su ranura y se conecta el cable a la Raspberry.
12) Se preparan las placas para integrarlas.
13) Se incorpora la placa pasando los espárragos por los agujeros, solo hay una posición posible y es la indicada en la imagen, además se conecta la placa de conexiones o Backplane
14) El espaciado de los conectores en el Backplane coincide con la posición de las placas apiladas por lo que solo hay una posición posible para conectarlo.
15) Se incorpora la base superior fijando tuercas para limitar el desplazamiento de esta.
16) Identificar las sujeciones laterales, los pasos 17 y 18 son para señalar las ranuras que determinar la posición de las sujeciones.

19) Se identifica el panel solar, el cable se pasa por dentro del satélite.
20) Se fija el panel solar en la ranura que posee la estructura y es conectado a la placa del subsistema de potencia.
21) Al accionar el interruptor del EPS el satélite se enciende, si no hubo fallas en la integración o componentes, la primera señal de que está funcionando es la luz verde parpadeante en la Raspberry y el adaptador WiFi.

C Código implementado a bordo del satélite

Esta sección corresponde al script que ejecuta Raspberry Pi al iniciar el sistema.

```
import json
import pickle
import csv
import os
import threading
import sys
import socket
import subprocess
from datetime import datetime, date
from time import sleep
from datetime import datetime
sys.path.append("/home/haise/Downloads/RPI-ADXL345")
import adxl345
from ina219 import INA219
from ina219 import DeviceRangeError
from mpu6050 import  mpu6050
import time
import board
import adafruit_mpu6050
IP_GS = '192.168.0.16'
#obtener directorio
path_now = os.path.realpath(os.path.dirname(__file__))
#CREACIÓN DE ARCHIVO DE REGISTRO
tm_dic_BASE = { #Keys para generar archivo de registro
                            "time": datetime.now(), #tiempo de toma de datos
                            "last_com":None, #ultimo comando recibido
                            "last_com_date": None, #Fecha ultimo comando recibido
                            "v5":0, # linea de 5 volts
                            "i5":0, #corriente en 5 volts
                            "p5":0, #potencia en 5 volts
                            "v3":0, #linea de 3.3 volts
                            "i3":0, #corriente en 3.3 volts
                            "p3":0, #potencia en 3.3V
                            "bat":0,# voltaje de batería
                            "sun":0, # voltaje de LDR
                            "acce_1_x":0,
                            "acce_1_y":0,
                            "acce_1_z":0,
                            "acce_2_x":0,
                            "acce_2_y":0,
                            "acce_2_z":0,
                            "gyro_X":0,
                            "gyro_Y":0,
                            "gyro_Z":0,
                            "temp":0
                            }
now = datetime.now()
current_time = now.strftime("%d_M%m_%Y-%H_m%M") #obtener fecha y hora
#crear nombre de archivo para registros
TM_file_new = path_now + f'/TM{current_time}.csv'
with open(TM_file_new, 'w', newline='') as f: # crea archivo
    writer = csv.writer(f)
    writer.writerow(tm_dic_BASE.keys())
```

```
def register_file_update(tm_dic): #actualiza el archivo
    with open(TM_file_new, 'a', newline='') as f2:
        writer2 = csv.writer(f2)
        writer2.writerow(tm_dic.values())
##########################################
class TELEMETRY():
    def __init__(self):
        self.TM_recorded = None
TM_RCRD = TELEMETRY()
class HAISE_state():
    def __init__(self,end_check_com,last_com, take_pic,linked):
        self.endCheck= end_check_com
        self.last_com = last_com
        self.v5line = 0
        self.i5line = 0
        self.p5line = 0
        self.v3line = 0
        self.i3line = 0
        self.p3line = 0
        self.batline = 0
        self.sunline = 0
        self.acce1x = 0
        self.acce1y = 0
        self.acce1z = 0
        self.acce2x = 0
        self.acce2y = 0
        self.acce2z = 0
        self.gyro_X = 0
        self.gyro_Y = 0
        self.gyro_Z = 0
        self.temp = 0
        self.TAKE_PIC = take_pic
        self.LINKED = linked
        self.ALIVE_FLAG=True
        self.ALIVE_SAT = True
        self.SEND_TM = False
        self.SEARCHING = True
        self.GS_FOUND = False
HS = HAISE_state("init",{
                            "command":"init",
                            "rec_date":"now"
                            },
                            take_pic=False,
                            linked = False
                            )
def clear():
        #os.system("clear")
        #print("\n\n\n")
        pass
def com_ss():
    global HS
    #socket telecomandos
    while True and HS.ALIVE_FLAG:
        if HS.GS_FOUND:
            try:
                bb = False
                attemps = 0
                while True:
                    try:
                        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                        #print("a")
                        s.connect((IP_GS, 4000))
                        try:
                            sleep(0.1)
                            s.settimeout(1.0)
                            receive = s.recv(1024)
                            s.settimeout(None)
                            print("Enlace establecido \n")
                            HS.LINKED=True
                            bb = True
                            break
                        except Exception as e:
                            s.close()
                            HS.LINKED = False
                            receive = False
                            if str(e) == "[Errno 104] Connection reset by peer":
                                HS.GS_FOUND = False
                                HS.SEARCHING = True
                                break
                            pass
                    except:
```

```
                        attemps +=1
                        sleep(0.1)
                        print(f"Attempting connection ... {attemps}")
                        if attemps > 10:
                            HS.GS_FOUND = False
                            HS.SEARCHING = True
                            break
                com_t = 0
                while receive and com_t!="end" and bb:
                    clear()
                    try:
                        receive = s.recv(1024)
                    except Exception as e:
                        HS.GS_FOUND = False
                        HS.SEARCHING = True
                        print("Estación terrestre perdida")
                        break
                    time = str(datetime.now())[0:-4]
                    com_t = receive.decode() # Recibir telecomando
                    HS.endCheck = com_t
                    if com_t != "end":
                            com_dic= {
                                    "command":com_t,
                                    "rec_date":time
                                    }
                            com_json = json.dumps(com_dic,indent = 1)
                            with open("coms.json","w") as updating:
                                    updating.write(com_json)
                            HS.last_com=com_dic
                            print(HS.last_com)
                            if com_t =="TAKE_PIC":
                                HS.TAKE_PIC=True
                            elif com_t == "KILL_OS":
                                HS.ALIVE_FLAG = False
                                print(HS.ALIVE_FLAG)
                                print("KILL")
                                break
                            elif com_t == "KILL_SAT":
                                HS.ALIVE_FLAG = False
                                HS.ALIVE_SAT = False
                                break
                            elif com_t == "GET_TM":
                                HS.SEND_TM = True
                    else:
                        print("Enlace cerrado")
                        HS.LINKED = False
                        s.close()
                    sleep(0.1)
                s.close()
                if com_t == "end":
                    s.close()
                HS.LINKED= False
                sleep(1)
            except Exception as e:
                #print(e)
                pass
        sleep(0.1)

def telemetry_update():
    global TM_RCRD,HS
    while True and HS.ALIVE_FLAG:
        tm_dic = {
                                "time": datetime.now().strftime("%d-%m-%Y %H:%M:%S.%f")[0:-3], #tiempo de toma de datos
now.strftime("%d_M%m_%Y-%H_m%M")
                                "last_com":HS.last_com["command"], #ultimo comando recibido
                                "last_com_date": HS.last_com["rec_date"],
                                "v5":HS.v5line, # linea de 5 volts
                                "i5":HS.i5line, #corriente 5V
                                "p5":HS.p5line, #potencia en 5V
                                "v3":HS.v3line, #linea de 3.3 volts
                                "i3":HS.i3line, #corriente 3.3V
                                "p3":HS.p3line, #potencia en 3.3V
                                "bat":HS.batline,# voltaje de batería
                                "sun":HS.sunline, # voltaje de LDR
                                # aceleraciones y velocidades angular de acce1=adxl345, acce2= mpu6050
                                "acce_1_x":HS.acce1x,
                                "acce_1_y":HS.acce1y,
                                "acce_1_z":HS.acce1z,
                                "acce_2_x":HS.acce2x,
                                "acce_2_y":HS.acce2y,
                                "acce_2_z":HS.acce2z,
```

```
                                "gyro_X":HS.gyro_X,
                                "gyro_Y":HS.gyro_Y,
                                "gyro_Z":HS.gyro_Z,
                                "temp":HS.temp
                                }
        TM_RCRD.TM_recorded = tm_dic
        register_file_update(tm_dic)
        sleep(5)

def TM_channel():
    global HS
    while True and HS.ALIVE_FLAG:
        if HS.GS_FOUND:
            try:
                cc = True
                attemps = 0
                if HS.LINKED:
                    #socket telemetría
                    s2 = socket.socket(socket.AF_INET,
                        socket.SOCK_STREAM)
                    while cc and attemps < 20:
                        try:
                            s2.connect((IP_GS, 4500))
                            cc = False
                        except:
                            attemps +=1
                            pass
                    while   not cc and HS.ALIVE_FLAG:
                        if not HS.LINKED:
                            break
                        tm_data = TM_RCRD.TM_recorded

                        coded_tm = pickle.dumps(tm_data)
                        s2.send(coded_tm) #Enviar telemetría
                        sleep(0.05)
                    s2.close()
                sleep(1)
            except Exception as e:
                #print(e)
                pass

def measure_ADXL345():
    global HS
    #Lectura del ADXL345
    accelerometer = adxl345.ADXL345(i2c_port=1, address=0x53)
    accelerometer.load_calib_value()
    accelerometer.set_data_rate(data_rate=adxl345.DataRate.R_100)
    accelerometer.set_range(g_range=adxl345.Range.G_16, full_res=True)
    accelerometer.measure_start()
    sensor = mpu6050(0x68) # Lecutra del  MPU6050
    i2c = board.I2C()
    mpu = adafruit_mpu6050.MPU6050(i2c)
    while True and HS.ALIVE_FLAG:
        x, y, z = accelerometer.get_3_axis_adjusted()
        x2, y2, z2 = mpu.acceleration
        gx , gy, gz = mpu.gyro
        temp2 = mpu.temperature
        HS.acce1x = str(x)[0:-13]
        HS.acce1y = str(y)[0:-13]
        HS.acce1z = str(z)[0:-13]
        accel_data = sensor.get_accel_data(g=True)
        gyro_data = sensor.get_gyro_data()
        temp = sensor.get_temp()
        HS.temp = str(temp)[0:-13]
        HS.acce2x = str(x2/9.807)[0:-9]
        HS.acce2y = str(y2/9.807)[0:-9]
        HS.acce2z = str(z2/9.807)[0:-9]
        HS.gyro_X = str(gx)[0:-13]
        HS.gyro_Y = str(gy)[0:-13]
        HS.gyro_Z = str(gz)[0:-13]
        sleep(0.1)

def measure_Power():
    global HS
    # identificadores
    # ad5v = 0x45
    # adbat = 0x40
    # ad_3v = 0x44
    # ad_sol = 0x41
    def read(ina):
        b_v = ina.voltage()
```

```
        try:
                b_c = ina.current()
                b_p = ina.power()
        except DeviceRangeError as e:
                print(e)
        return b_v, b_c, b_p
    SHUNT_OHMS = 0.1
    max_AMP = 1.0
    ad5v = 0x40
    adbat = 0x41
    ad_3v = 0x45
    ad_sol = 0x44
    ina_5v = INA219(SHUNT_OHMS,max_AMP,address=ad5v)
    ina_5v.configure(ina_5v.RANGE_16V)
    ina_bat = INA219(SHUNT_OHMS,max_AMP,address=adbat)
    ina_bat.configure(ina_bat.RANGE_16V)
    ina_3v = INA219(SHUNT_OHMS,max_AMP,address=ad_3v)
    ina_3v.configure(ina_3v.RANGE_16V)
    ina_sol = INA219(SHUNT_OHMS,max_AMP,address=ad_sol)
    ina_sol.configure(ina_bat.RANGE_16V)
    while True and HS.ALIVE_FLAG:
        V5_line = read(ina_5v)
        bat_line =read(ina_bat)
        v3_line =read(ina_3v)
        sol_line = read(ina_sol)
        HS.v5line = V5_line[0]
        HS.i5line = str(V5_line[1])[0:-11]
        HS.p5line = str(V5_line[2])[0:-11]
        HS.batline = bat_line[0]
        bat_i = str(bat_line[1])[0:-11]
        bat_p = str(bat_line[2])[0:-11]
        HS.v3line = v3_line[0]
        HS.i3line = str(v3_line[1])[0:-11]
        HS.p3line = str(v3_line[2])[0:-11]
        HS.sunline = sol_line[0]
        if bat_line[0]< 3.2:
            HS.ALIVE_FLAG = False
            HS.ALIVE_SAT = False
        sleep(0.2)
    pass

#socket imagenes
def take_pic():
    global HS
    while True and HS.ALIVE_FLAG:
        if HS.GS_FOUND:
            try:
                while HS.endCheck!="end" and HS.LINKED and HS.ALIVE_FLAG:
                    if HS.TAKE_PIC:
                        subprocess.run(["raspistill","-rot","270","-o","image.jpg","-w","1920","-h","1080"])
                        s3 = socket.socket(socket.AF_INET,
                                socket.SOCK_STREAM)
                        dd = True
                        attemps = 0
                        while dd and attemps < 20:
                            try:
                                s3.connect((IP_GS, 3000))

                                dd = False
                            except:
                                attemps +=1
                                #print(attemps)
                                pass
                        #im = open("c:/Users/Robocop/Desktop/HAISE/imagen.jpg","rb")
                        im = open("image.jpg","rb")
                        for i in im:
                            s3.send(i)
                        HS.TAKE_PIC=False
                        #print("image sent")
                        sleep(0.5)
                        s3.close()
                sleep(1)
            except Exception as e:
                #print(e)
                pass
        sleep(0.1)

# Enviar toda la telemetría registrada desde que se encendió el satelite.
def send_all_TM():
    global HS
    while True and HS.ALIVE_FLAG:
```

```
        if HS.GS_FOUND:
            try:
                cc = True
                attemps = 0
                if HS.SEND_TM and HS.LINKED:
                    #socket telemetría
                    s_tm = socket.socket(socket.AF_INET,
                        socket.SOCK_STREAM)
                    while cc and attemps < 20:
                        try:
                            s_tm.connect((IP_GS,7000))
                            cc=False
                        except Exception as e:
                            attemps +=1
                            #print (e)
                            if attemps>20:
                                HS.SEND_TM = False
                    with open(TM_file_new, "r") as f:
                        reader = csv.reader(f)
                        for i, line in enumerate(reader):
                            s_tm.send(pickle.dumps(line))
                    s_tm.close()
                    HS.SEND_TM = False
            except Exception as e:
                #print(e)
                pass
            sleep(1)
        sleep(0.1)

def try_something():
    global IP_GS, addr
    while True and HS.ALIVE_FLAG:
        if HS.SEARCHING:
            try:
                print("Buscando estación terresre")
                socket_GS = socket.socket(socket.AF_INET,
                                socket.SOCK_STREAM)
                socket_GS.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,1)
                print(1)
                socket_GS.bind(('', 8200))
                print(2)
                socket_GS.listen(1)
                print(3)
                d, addr = socket_GS.accept()
                print(4)
                IP_GS = addr[0]
                telem_1 = d.recv(2048)
                socket_GS.close()
                HS.SEARCHING = False
                HS.GS_FOUND = True
                print("Estación terrestre encontrada")
            except Exception as e:
                print("Error try_something", e)
                socket_GS.close()
                pass
        sleep(1)

if __name__== "__main__":
    #Threads para multiprocesos
    t_coms = threading.Thread(target=com_ss)
    t_coms.daemon = True

    t_tm_up = threading.Thread(target=telemetry_update)
    t_tm_up.daemon = True

    t_tm_send = threading.Thread(target=TM_channel)
    t_tm_send.daemon = True

    t_camera = threading.Thread(target=take_pic)
    t_camera.daemon = True

    t_all_TM = threading.Thread(target=send_all_TM)
    t_all_TM.daemon = True

    t_power = threading.Thread(target=measure_Power)
    t_power.daemon = True
    t_ADXL = threading.Thread(target=measure_ADXL345)
    t_ADXL.daemon = True
    t_searching = threading.Thread(target = try_something)
    t_searching.daemon = True
    t_searching.start()
```

```
    sleep(0.2)
    t_coms.start()
    sleep(0.2)
    t_tm_up.start()
    t_tm_send.start()
    t_camera.start()
    t_all_TM.start()
    t_power.start()
    t_ADXL.start()
    t_searching.join()
    sleep(0.2)
    t_coms.join()
    sleep(0.2)
    t_tm_up.join()
    t_tm_send.join()
    t_camera.join()
    t_all_TM.join()
    t_power.join()
    t_ADXL.join()
    if not HS.ALIVE_SAT:
        print("Power Off")
        sleep(5)
        os.system("sudo shutdown now")
    sys.exit()
```

Adicional al script del *Flight Software*, requiere otro archive para calibrar uno de los acelerómetros.

```
import adxl345
import time
import os
def clear():
    os.system("clear")

accelerometer = adxl345.ADXL345(i2c_port=1, address=0x53)
accelerometer.load_calib_value()
accelerometer.set_data_rate(data_rate=adxl345.DataRate.R_100)
accelerometer.set_range(g_range=adxl345.Range.G_16, full_res=True)
accelerometer.measure_start()

accelerometer.calibrate()   # Calibra una vez

n = 0
while(True) and n<5:
    clear()
    n = n+0.25
    x, y, z = accelerometer.get_3_axis_adjusted()
    print('x: ', x, 'y: ', y, 'z: ', z,"\n")#
    print('pitch: ',"\n", accelerometer.get_pitch())


    time.sleep(0.25)
```

Para instalar la librería del adxl345 es necesario seguir las instrucciones detalladas en:

https://github.com/utthawut/RPI-ADXL345

D Código implementado para la estación de control

```
from time import sleep
import os
import json
import copy
import socket
import threading
import pickle
import csv
from datetime import datetime
import sys

# Import the required libraries
from tkinter import *
from tkinter import ttk, messagebox
from PIL import Image, ImageTk
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

import tkinter

def try_something(): #Función para iniciar conexiones con satélite.

    attempt = 0
    flag = True
    while flag and attempt<200:
        try:
            s2 = socket.socket(socket.AF_INET,
                               socket.SOCK_STREAM)
            s2.connect(('haisesat.local', 8200))
            s2.send(bytes("coded_tm","UTF-8"))
            s2.close()
            flag = False
        except:
            s2.close()
            attempt +=1
            pass

try_something()

path_now = os.path.realpath(os.path.dirname(__file__))

class GS_state:
    def __init__(self, telem):
        self.com_sent = "---"
        self.com_sent_date = "---"
        self.last_com_input = "---"
        self.com_count = 0
        self.error_inner = None
        self.telemetry = telem
        self.TAKE_PIC = False
        self.LINKED = False
        self.IOstate = False
        self.TM_receiving = False
        self.working = True
        self.UNLINK_UPDATE = False

"""funciones generales para ejecución"""
# Limpia ventana de comandos de windows
def clear():
    #os.system("cls")
    pass
#Guarda comando ingresado en un txt para historico, y en json para comparar con el siguiente comando
def com_in_update(com,t,N):
    """
    com: comando ingresado para enviar, type: str
    t: tiempo actual, type: str
    N: contador de comandos, type: int
    """
    dict = {"com_num":N,"command":com,"com_date":t}
    json_dic = json.dumps(dict,indent=1)
    with open("command_now.json","w") as update:
        update.write(json_dic)
    with open("command_hist.txt","a") as hist:
        hist.write(str(dict) + "\n")
    return dict
```

```
# Abre archivo de comandos historico para tener el contador de comandos, si no existe, el contador es 0
try:
    with open("command_hist.txt", "rb") as file:
        try:
            file.seek(-2, os.SEEK_END)
            while file.read(1) != b'\n':
                file.seek(-2, os.SEEK_CUR)
        except OSError:
                file.seek(0)
        last_line = file.readline().decode("UTF-8")
        alpha = json.loads(last_line.__str__().replace("'",'"')) #Asegura que el diccionario tenga la sintaxis correcta
        N = alpha["com_num"] # encuentra el contador del ultimo comando enviado de una sesión anterior
except OSError:
    print(OSError)
    print("Contador en 0 \n")
    N=0

# Inicia la sesión de trabajo con el contador de comandos enviados en 0 o el encontrado en el archivo command_hist.txt
com_in_update("Init",str(datetime.now())[0:-4],N)
tm_dic = { #Keys para generar archivo de registro
                            "time": datetime.now(), #tiempo de toma de datos
                            "last_com":"None", #ultimo comando recibido
                            "last_com_date": "None", #Fecha ultimo comando recibido
                            "v5":0, # linea de 5 volts
                            "i5":0, #corriente en 5 volts
                            "p5":0, #potencia en 5 volts
                            "v3":0, #linea de 3.3 volts
                            "i3":0, #corriente en 3.3 volts
                            "p3":0, #potencia en 3.3V
                            "bat":0,# voltaje de batería
                            "sun":0, # voltaje de LDR
                            "acce_1_x":0,
                            "acce_1_y":0,
                            "acce_1_z":0,
                            "acce_2_x":0,
                            "acce_2_y":0,
                            "acce_2_z":0,
                            "gyro_X":0,
                            "gyro_Y":0,
                            "gyro_Z":0,
                            "temp":0
                            }
GS = GS_state(telem=tm_dic)

def TM_channel():#Canal de telemetría constante
    global GS, com_recv_label,com_recv_date_label, v5_label,v3_label,bat_label,sun_label
    sleep(1)
    while True and GS.working:
        socket_tm2 = socket.socket(socket.AF_INET,
                    socket.SOCK_STREAM)
        socket_tm2.bind(('', 4500))
        socket_tm2.listen(1)
        d, addr = socket_tm2.accept()

        while GS.LINKED:
            try:

                telem_1 = d.recv(2048)

                TM_RCV= pickle.loads(telem_1)
                #print(telem_1)
                GS.telemetry = TM_RCV
            except Exception as e:
                pass
        socket_tm2.close()
        sleep(1)

def TM_all():#REcepción de telemetría completa
    while True and GS.working:
        if GS.TM_receiving and GS.LINKED:
            socket_tm = socket.socket(socket.AF_INET,
                    socket.SOCK_STREAM)
            socket_tm.bind(('', 7000))
            socket_tm.listen(1)
            d2, addr = socket_tm.accept()
            try:
                line=True
                lista = []
                while line:
                    line = d2.recv(2048)
```

```
                    if line == b'':
                        break
                    else:
                        try:
                            data = pickle.loads(line)
                            print(data)
                        except:
                            pass
                    lista.append(data)
                    with open (path_now + "/telemetria.csv", 'w', newline='') as csv_file:
                        csv_writer = csv.writer(csv_file)
                        for row in lista:
                            csv_writer.writerow(row)
            except Exception as e:
                print(e)
            socket_tm.close()
        sleep(1)
def photo_CH():#FUNCION PARA ABRIR CANAL DE DESCARGA DE IMAGENES
    global GS
    while True and GS.working:
            while GS.last_com_input!= "end" and GS.LINKED:
                if GS.TAKE_PIC:
                    socket3 = socket.socket(socket.AF_INET,
                                            socket.SOCK_STREAM)
                    socket3.bind(("", 3000))
                    socket3.listen(1)
                    ss, address = socket3.accept()
                    im = path_now + "/imagen.jpg"
                    condition = True
                    f = open(im, "wb")
                    print("imagen creada")
                    while condition:
                        image = ss.recv(2048)
                        f.write(image)
                        if str(image) == "b''":
                            condition = False
                    GS.TAKE_PIC=False
                    socket3.close()
                sleep(0.5)

def com_sender():#FUNCION PARA ENVIO DE TELECOMANDOS, ENVIA EL ULTIMO TELECOMANDO INGRESADO
    global GS, win, com_recv_label,com_recv_date_label,com_sent_label,com_sent_date_label, v5_label,
v3_label,bat_label,sun_label
    ##########
    while True and GS.working:
        socket1 = socket.socket(socket.AF_INET,
                            socket.SOCK_STREAM)
        socket1.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        """Inicia lectura de JSON para guardar el com_num inicial"""
        socket1.bind(('',4000))
        socket1.listen(1)
        # sleep(1)
        c, addr = socket1.accept()

        if GS.IOstate:
            def update_label():
                global com_recv_label,com_recv_date_label,GS
                com_sent_label["text"] = "Último TC Enviado: " + GS.com_sent
                com_sent_date_label["text"] = "Fecha Enviado: " + GS.com_sent_date

            a=True
            while a :
                try :
                    with open("command_now.json", "r") as file:
                        command_state_old = json.load(file)
                        com_num_old  = command_state_old["com_num"]
                        a = False
                except OSError:
                    print(OSError)
                    pass
            ###################################
            """Inicia server para comunicaciones con el satelite"""

            c.send(b"Initiated")
            GS.LINKED = True

            ###################################
            sleep(1) #espera 1segundo
            ########################################
            """Espera por cambios en el JSON para enviar dicho comando"""
```

```
            a = True
            #and GS.last_com_input !="end"
            while a :
                with open("command_now.json", "r") as file:
                        try:
                            command_state = json.load(file)
                            command  = command_state["command"]
                            com_num  = command_state["com_num"]
                        except:
                            command = "end"
                            com_num = copy.deepcopy(com_num_old)
                            pass

                        sleep(0.05)
                        if (com_num_old != com_num) and (command !="end"):
                            com_num_old = copy.deepcopy(com_num)

                            c.send(command.encode())  # Enviar telecomando
                            time_send = str(datetime.now())[0:-4]
                            com_sent = f"{command}; #: {com_num}"
                            GS.com_sent = com_sent
                            GS.com_sent_date = time_send
                            update_label()
                            if command == "KILL_OS" or command == "KILL_SAT":
                                GS.LINKED = False
                        elif (com_num_old != com_num) and (command =="end"):

                            c.send("end".encode())
                            a = False
                            GS.IOstate=False
                            GS.LINKED = False
                            print("IOState apagado")
                            sleep(1)
                            # win.quit()
                            #sys.exit()

        socket1.close()


def user_gui():# ABRE INTERFAZ DE USUARIO EN UNA VENTANA
    global GS, com_recv_label, win, canvas1, N, com_recv_date_label, com_sent_label,com_sent_date_label, v5_label,
v3_label,bat_label,sun_label
    global acce1x_label, acce1y_label, acce1z_label, acce2x_label, acce2y_label, acce2z_label, gyro_X_label, gyro_Y_label,
gyro_Z_label, link_label, temp_label
    win=Tk()
    win.title("Estación terrestre")
    win.geometry("1500x350")
    icon = Image.open(path_now + "/escudo45x55.bmp")
    icon = ImageTk.PhotoImage(icon)
    win.wm_iconphoto(False,icon)
    win_title = Label(win, text='Interfaz Estación Terrestre',font=('Aerial 18'))
    com_line_input = Entry(win, width = 20)
    com_list = ["end",  # Finaliza loop de ingreso de comandos
                "GET_TM",  # Recibir desde sat Telemetría
                "TAKE_PIC",  # Tomar foto y enviarla
                "GO_SAFE",  # Ir a modo seguro y reinicia contador de telecomandos
                "RECOVER",  # Volver a modo Normal
                "ALL_OK", # Estado de sensores OK
                "KILL_SAT", # Apaga el sistema del satelite ¡
                "KILL_OS" # Apaga el software del satelite ¡
                ]
    #PREPARA EL TELECOMANDO PARA SER ENVIADO

    def on_closing():
        if messagebox.askokcancel("Quit","Do you wand to quit?"):
            GS.working = False
            sys.exit()
    win.protocol("WM_DELETE_WINDOW", on_closing)



    def send_TC():
        global N
        command = com_line_input.get()
        if command == "init":
            GS.IOstate = True
        elif command in com_list and GS.IOstate:
            if command == "GO_SAFE":
                N = 0
            elif command == "TAKE_PIC":
```

```
            N = N+1
            GS.TAKE_PIC=True
        elif command == "GET_TM":
            N = N+1
            GS.TM_receiving = True

        else:
            N = N + 1
        t = str(datetime.now())[0:-4]
        dictio = com_in_update(command, t, N)
        GS.last_com_input = command
        sleep(0.2)
    else:
        com_sent_label["text"] = f"Comando erroneo: {command}"
    com_line_input.delete(first=0, last=30)
    print(GS.IOstate)

#Telecomandos
send_com_B = Button(win, text="Send Command", command=send_TC, font=('Helvetica bold', 10), justify="right")
link_label = Label(win, text = "---", font =('Helvetica bold',12),bg='#fff', fg='#f31f00', justify="right")
com_sent_label = Label(win, text="Último TC enviado: ---", font=('Aerial 18'), justify="right")
com_sent_date_label = Label(win, text="Fecha envío: ---", font=('Aerial 18'), justify="right")
com_recv_label = Label(win, text="Último TC recibido: ---", font=('Aerial 18'), justify="right")
com_recv_date_label = Label(win, text="Fecha recibido: ---", font=('Aerial 18'), justify="right")
#Potencia
v5_label = Label(win, text="5V: ---", font=('Aerial 18'),justify="left")
v3_label = Label(win, text="3.3V: ---", font=('Aerial 18'),justify="left")
bat_label = Label(win, text="Bat: ---", font=('Aerial 18'), justify="right")
sun_label = Label(win, text="Sol: ---", font=('Aerial 18'), justify="right")

temp_label = Label(win,text="Temperatura: ---", font=('Aerial 18'), justify="right")
#ADCS
acce1x_label = Label(win, text="X: ---", font=('Aerial 18'), justify="right")
acce1y_label = Label(win, text="Y:---", font=('Aerial 18'), justify="right")
acce1z_label= Label(win, text="Z:---", font=('Aerial 18'), justify="right")
acce2x_label = Label(win, text="X ---", font=('Aerial 18'), justify="right")
acce2y_label = Label(win, text="Y---", font=('Aerial 18'), justify="right")
acce2z_label= Label(win, text="Z---", font=('Aerial 18'), justify="right")
gyro_X_label = Label(win, text="X---", font=('Aerial 18'), justify="right")
gyro_Y_label = Label(win, text="Y---", font=('Aerial 18'), justify="right")
gyro_Z_label = Label(win, text="Z---", font=('Aerial 18'), justify="right")
#titulos sensores
acce1_title = Label(win, text="ADXL345 \n Acelerómetro", font=('Aerial 18'), justify="right")
acce2_title = Label(win, text="MPU-6050 \n Acelérometro", font=('Aerial 18'), justify="right")
gyro_title = Label(win, text="Giroscopio", font=('Aerial 18'), justify="right")
gyro_X_label = Label(win, text=" ------1", font=('Aerial 18'), justify="right")
gyro_Y_label = Label(win, text=" ------1", font=('Aerial 18'), justify="right")
gyro_Z_label = Label(win, text=" ------1", font=('Aerial 18'), justify="right")

win_title.grid(row=0,column=1)
#Telecomandos
com_line_input.grid(row=1,column=1)
send_com_B.grid(row=1,column=2)
link_label.grid(row=1,column=3)
com_sent_label.grid(row=2,column=1)
com_sent_date_label.grid(row=2,column=3)
com_recv_label.grid(row=3,column=1)
com_recv_date_label.grid(row=3,column=3)
#Potencia
v5_label.grid(row=5,column=1)
v3_label.grid(row=6,column=1)
bat_label.grid(row=7,column=1)
sun_label.grid(row=8, column=1)
temp_label.grid(row=1, column= 5)
#ADCS
acce1_title.grid(row=5, column=3)
acce2_title.grid(row=5, column=5)
gyro_title.grid(row=5, column=7)

acce1x_label.grid(row=6,column=3)
acce1y_label.grid(row=7,column=3)
acce1z_label.grid(row=8,column=3)
acce2x_label.grid(row=6,column=5)
acce2y_label.grid(row=7,column=5)
acce2z_label.grid(row=8,column=5)
gyro_X_label.grid(row=6,column=7)
gyro_Y_label.grid(row=7,column=7)
gyro_Z_label.grid(row=8,column=7)

# MOSTRAR FOTO EN UNA VENTANA NUEVA
def photo_show():
```

```
        try:
            img_window = Toplevel(win)
            img_window.title("Imagen recibida")
            img_window.wm_iconphoto(False, icon)
            image_recv = Image.open(path_now + "/imagen.jpg")
            w,h = image_recv.size
            img_window.geometry(f"{int(w/2)}x{int(h/2)}")
            image_recv = image_recv.resize((int(w/2), int(h/2)))
            try:
                test = ImageTk.PhotoImage(image_recv)
                label1 = Label(img_window, image=test)
                label1.image = test
                label1.place(x= 0, y = 0)
            except Exception as bn:
                print(bn)
            # Position image
        except Exception as e:
            print(e)
            pass
    pic_button = Button(win, text="show photo", command=photo_show, font=('Helvetica bold', 10))
    pic_button.grid(row=10,column=1)
    win.mainloop()

def gui_update():
    global GS
    global com_recv_label,com_recv_date_label, v5_label,v3_label,bat_label,sun_label
    global acce1x_label, acce1y_label, acce1z_label,acce2x_label,acce2y_label,acce2z_label
    global gyro_Y_label, gyro_Z_label, gyro_X_label, link_label, temp_label

    while True and GS.working:

        if GS.LINKED:

            try:

                link_label["text"] = "LINKED"
                link_label.config(bg='#fff', fg='#000fff000')
                #potencia

                if GS.telemetry["bat"] >= 4.0:

                    bat_label.config(fg='#000fff000')
                elif  GS.telemetry["bat"] >= 3.7 and  GS.telemetry["v5"] < 4.0:

                    bat_label.config( fg='#99ff66')
                elif GS.telemetry["bat"] >= 3.4 and  GS.telemetry["v5"] < 3.7:
                    bat_label.config( fg='#ff6600')
                else:
                    bat_label.config( fg='#ff0000')

                v5_label["text"] = "5V: " + str(GS.telemetry["v5"]) + " V"
                v3_label["text"] = "3.3V: " + str(GS.telemetry["v3"]) + " V"
                bat_label["text"] = "Bat: " + str(GS.telemetry["bat"]) + " V"
                sun_label["text"] = "Sol: " + str(GS.telemetry["sun"]) + " V"
                #ADCS
                acce1x_label["text"] = "X: " + str(GS.telemetry["acce_1_x"])+ " g"
                acce1y_label["text"] = "Y: " + str(GS.telemetry["acce_1_y"]) + " g"
                acce1z_label["text"] = "Z: " + str(GS.telemetry["acce_1_z"])  + " g"
                acce2x_label["text"] = "X: " + str(GS.telemetry["acce_2_x"])  + " g"
                acce2y_label["text"] = "Y: " + str(GS.telemetry["acce_2_y"])  + " g"
                acce2z_label["text"] = "Z: " + str(GS.telemetry["acce_2_z"])  + " g"
                gyro_X_label["text"] = "X: " + str(GS.telemetry["gyro_X"])  + " °/s"
                gyro_Y_label["text"] = "Y: " + str(GS.telemetry["gyro_Y"])  + " °/s"
                gyro_Z_label["text"] = "Z: " + str(GS.telemetry["gyro_Z"])  + " °/s"
                temp_label["text"] = "Temperatura:" + str(GS.telemetry["temp"]) + "C°"
                # telecomando

                com_recv_label["text"] = "Último TC recibido: " + GS.telemetry["last_com"]
                com_recv_date_label["text"] = "Fecha recepción: " + GS.telemetry["last_com_date"]
            except Exception as e:
                print(e)
                pass
            GS.UNLINK_UPDATE = True
            sleep(0.05)


        elif not GS.LINKED and GS.UNLINK_UPDATE:
            try:
                link_label["text"] = "---"
                link_label.config(bg='#fff', fg='#f31f00')
                v5_label["text"] = "5V: " + "-"
```

```
                v3_label["text"] = "3.3V: "+ "-"
                bat_label["text"] = "Bat: " + "-"
                sun_label["text"] = "Sol: " + "-"
                #ADCS
                acce1x_label["text"] = "X: " + "-"
                acce1y_label["text"] = "Y: " + "-"
                acce1z_label["text"] = "Z: " + "-"
                acce2x_label["text"] = "X: " + "-"
                acce2y_label["text"] = "Y: " + "-"
                acce2z_label["text"] = "Z: " + "-"
                gyro_X_label["text"] = "X: " + "-"
                gyro_Y_label["text"] = "Y: " + "-"
                gyro_Z_label["text"] = "Z: " + "-"
            except:
                pass
            GS.UNLINK_UPDATE = False



# INICIA EL PROGRAMA DE LA ESTACIÓN DE CONTROL, EJECUTANDO TAREAS EN PARALELO
if __name__== "__main__":

    t3 = threading.Thread(target=user_gui)
    t3.daemon = True

    t2 = threading.Thread(target=com_sender)
    t2.daemon = True

    t4 = threading.Thread(target=TM_channel)
    t4.daemon = True
    t5 = threading.Thread(target=photo_CH)
    t5.daemon = True

    t_tm_all = threading.Thread(target=TM_all)
    t_tm_all.daemon = True

    t_gui_up = threading.Thread(target=gui_update)
    t_gui_up.daemon=True
    t3.start()
    sleep(0.5)
    t2.start()

    t4.start()
    t5.start()
    t_tm_all.start()
    t_gui_up.start()

    t3.join()
    sleep(0.5)
    t2.join()

    t4.join()
    t5.join()
    t_tm_all.join()
    t_gui_up.start()
    sys.exit()
```

E Pasos iniciales para instalación de software de vuelo

E.1 Instalación de Sistema operativo.

Instrucciones para instalación desde cero:

- Descargar imagen de RASPBIAN BULLSEYE 11.
- Descargar https://www.raspberrypi.com/software/ y montar imagen en tarjeta SD de 16GB o más

- Habilitar SSH y definir Usuario:Hostname:Contraseña en software de instalación.
- Configurar red local WiFi que se usará.

E.2 Actualización de sistema e instalación de librerías

- Insertar microSD en RPZ.
- Conectar HUB USB con teclado y Smartphone que permita compartir internet via USB.
- Conectar alimentación de RPZ y luego esperar boot de sistema.

Una vez haya iniciado el sistema y entre a la línea de comandos, verificar que hay acceso a internet ingresando en la línea de comando de Raspberry:

```
ifconfig -a
```

Debe verse el tag `usb0` y tener una ip asignada en el tag `inet`

Con el acceso a internet confirmado, ejecutar las siguientes líneas

```
sudo apt-get update && sudo apt-get install -f
```

Esto buscará e instalará actualizaciones para el sistema operativo.

Luego se instalarán paquetes necesarios para la instalación de los drivers

```
sudo apt-get install -y build-essential git raspberrypi-kernel-headers
```

Otra línea a ejecutar es la siguiente:

```
sudo apt-get dist-upgrade -f
```

Se ingresa “Y” confirmando que aceptamos la instalación. Esto tomara varios minutos así que solo queda esperar.

Finalmente se reiniciará el sistema usando `sudo reboot`

E.3 Descarga e instalación de driver de TP-Link WN725N si se cuenta con una Raspberry Pi Zero sin WiFi y se usa el Wifi USB de TPlink.

Cambiaremos el directorio en el cual descargaremos los archivos con el comando `cd Downloads`, luego ejecutar la siguiente línea:

```
git clone https://github.com/lwfinger/rtl8188eu.git
```

Esto lo que hará será clonar los archivos de este git a la carpeta en la cual nos encontramos.

Luego ejecutaremos la siguiente línea: `cd rtl8188eu`, que nos cambiará el directorio a la carpeta descarga, acto seguido ejecutar `make`, esto nuevamente tomará bastante tiempo. Finalizada esto ejecutar: `sudo make install`, tomará bastante tiempo por lo que solo queda esperar. Una vez finalizada la tarea anterior, ingresar: `sudo reboot`, para aplicar los cambios, una vez iniciado, apagar el sistema con: `sudo shutdown now`.

Conectar el WIFI USB y prender la Raspberry.

E.4 Configurar WIFI

En este paso se tiene que editar dos archivos del sistema. Primero ejecutaremos:

```
sudo nano /etc/network/interfaces
```

Dejar el archivo tal como se indica a continuación:

```
source /etc/network/interfaces.d/*
auto lo
iface lo inet loopback
auto wlan0
allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
```

El siguiente archivo para editar se realiza con la siguiente línea:

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

Debe contener lo siguiente

```
country = CL
update_config=1
ctrl_interfaces=DIR=/var/run/wpa_supplicant GROUP=netdev
network={
        scan_ssid=1
        ssid="Nombre red WIFI"
        psk="Contraseña red WIFI"
}
```

Hecho esto se reinicia el sistema con `sudo reboot` para hacer efectivos los cambios y una vez que inicie el sistema, la Raspberry debería estar conectada a la red que configuramos cada vez que inicie.

Para el caso de una Raspberry Pi Zero W basta con configurar la red en la instalación del sistema operativo en la microSD con el programa que provee el equipo de Raspberry Pi Foundation.

Más detalles sobre configurar la red WiFi en una Raspberry Pi y solución de errores con las credenciales de acceso para la RPZ consultar los siguientes foros:

https://gist.github.com/MBing/de297a8ae5e8a191c55a67a568d20d31#file-install_wlan_dongle-sh-L51

https://www.electronicshub.org/setup-wifi-raspberry-pi-2-using-usb-dongle/

https://docs.bluehosting.cl/troubleshooting/servidores/como-solucionar-el-mensaje-de-error-la-clave-del-equipo-remoto-ha-cambiado-al-iniciar-sesion-via-ssh.html

F Verificación de Requisitos y Restricciones

Se utilizaron los métodos descritos en la Tabla 3-5 y algunos resultados se presentan a continuación:

F.1 Medición de la autonomía del HAISE-Sat.

Se utilizaron dos modos de operación para medir la autonomía del HAISE-Sat:

- Modo 1: El primero corresponde a encenderlo con una batería completamente cargada y registrar con los sensores, el voltaje de la batería hasta que se apaga por falta de energía.
- Modo 2: Corresponde a programar en el software de vuelo una instrucción para que se apague de forma automática si el voltaje de la batería es inferior a 3.2V

Los resultados de esto se observan en las Figura 7-8 y Figura 7-9, se verifica que la autonomía del HAISE-Sat supera las 2 horas establecidas como requisito en el diseño. Limitar la descarga de la batería permite proteger la integridad de esta en el tiempo.

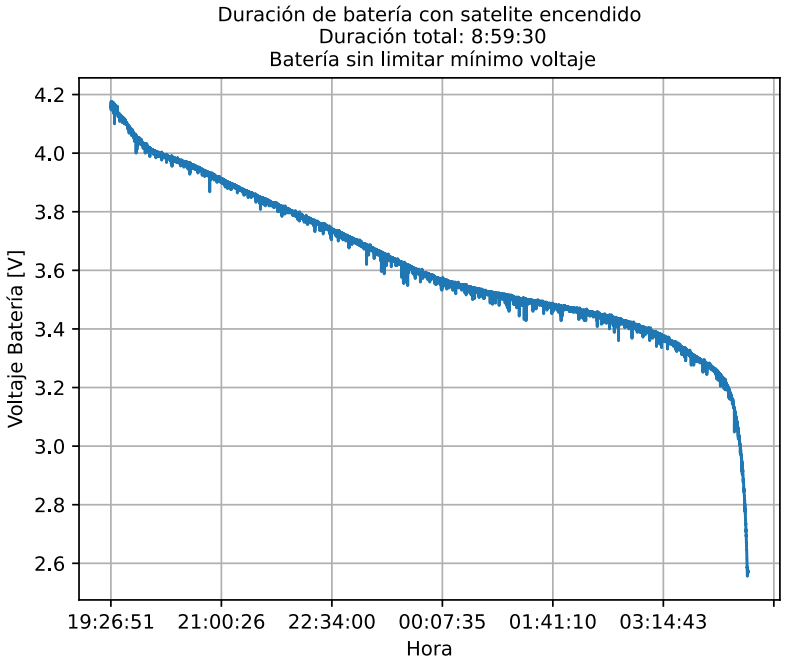


Figura 7-8. Estado de la batería en el Modo 1.

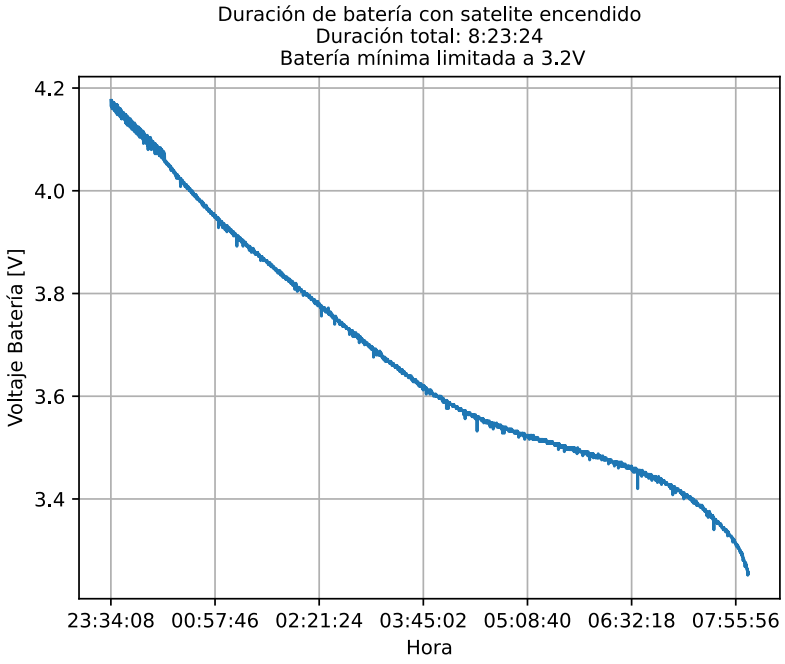
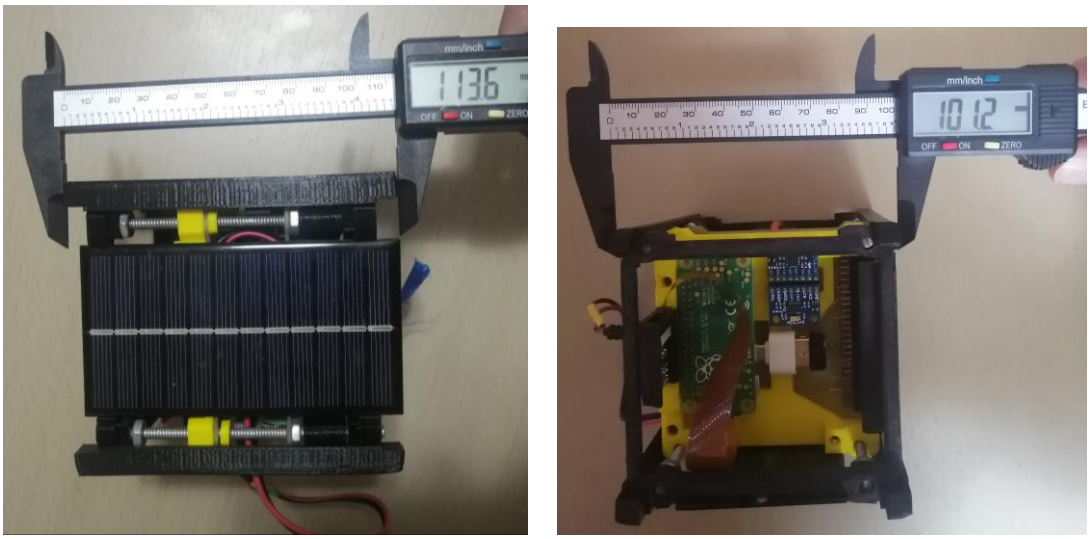


Figura 7-9. Estado de la batería en el Modo 2.

F.2 Medición de dimensiones en el volumen.

Las dimensiones indicadas por el estándar CubeSat corresponden a 100x100x113.5 mm^3, siendo los 113.5 mm al largo de las sujeciones laterales (medido en la Figura 7-10.a)), y una base de 10 mm x 10 mm (medido en las figuras Figura 7-10.b y Figura 7-10.c). Las dimensiones logradas en la fabricación caen dentro del 2% de error, que no compromete sus capacidades como herramienta de entrenamiento. Respetar el estándar sigue siendo un desafío y puede ser un parámetro de estudio en futuros trabajos para lograr las tolerancias deseadas al conocer bien el proceso de fabricación.



a) b)

c)

Figura 7-10. Dimensiones del CubeSat Bus fabricado.

F.3 Medición de masa

La masa medida del HAISE-Sat es de 401 gramos completamente integrado, esto da margen para implementar mejoras que conlleven a un incremento en la masa, recordando que el límite para un CubeSat 1U está en los 2 kg según la revisión 14 del *CubeSat Design Specification* (Cal Poly, 2020)

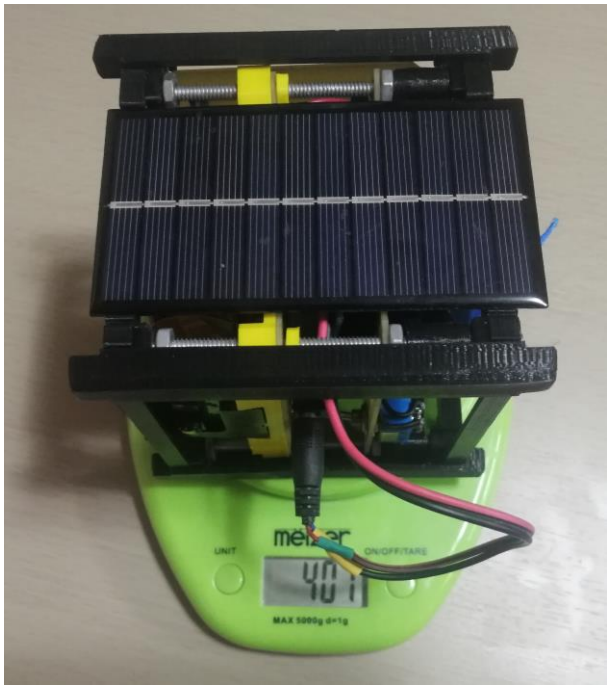


Figura 7-11. Masa del HAISE-Sat medida con una balanza de cocina.