UNIVERSIDAD DE CONCEPCIÓN

FACULTAD DE INGENIERÍA

DEPARTAMENTO INGENIERÍA MECÁNICA

DISEÑO DE UN SIMULADOR DE DESEMPEÑO DE APUNTAMIENTO DE CARGAS ÚTILES DE OBSERVACIÓN TERRESTRE ÓPTICAS EN *CUBESAT*S EN ÓRBITAS BAJAS

POR

<Matías Ignacio Tacul Vargas>

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

Profesores Guía:
<PhD, Bernardo Andrés Hernández Vicente>
<PhD (C), Alejandro Ignacio López Telgie>

Marzo 2024
Concepción (Chile)

Esta memoria está dedicada a todos aquellos que aportaron con su conocimiento y apoyo a la realización de este trabajo.

En primera instancia quiero agradecer a mi familia, Cecilia, Cristopher, Carla, Ramon, Alicia y Joaquín, por apoyarme con la decisión de venir a Concepción desde Coyhaique para poder estudiar la carrera que en ese momento me llamo la atención, además de brindarme el sustento tanto económico como emocional y por siempre apoyarme en cada momento importante en mi vida. Los quiero mucho

También agradecer a mi novia Camila y sus padres Erica y Manuel, que son mi segunda familia y me ayudaron una inmensidad durante estos 6 años en Concepción. Lograron empujarme a seguir adelante siempre, además de estar ahí para aconsejarme y entregarme el cariño necesario para sobrellevar la carga académica y los momentos difíciles durante mi estadía en la universidad. Gracias sobre todo a Camila que me ayudo a ser una mejor persona y a entregarme la confianza para expresar mis problemas y desahogarme para seguir avanzando en la vida. Estoy eternamente agradecido con ustedes y los quiero mucho.

Finalmente, agradecer a todas aquellas personas que conocí durante mi estadía universitaria y que fueron un apoyo para seguir avanzando en la carrera. A mi profesor guía Bernardo Hernández por su apoyo y sabiduría entregada durante este último año en que trabajamos juntos para el desarrollo de este trabajo y a mi co-guía Alejandro López que ayudó con su conocimiento a destrabar el proyecto con sus ideas y recomendaciones. Al grupo de trabajo conformado por Álvaro, Geovanni y German, en los cuales se forjó un buen equipo en el cual nos apoyamos mutuamente. A todos los compañeros de la carrera que hicieron la estadía en la universidad más grata. También a mis compañeros de magister por la buena onda durante estos últimos 2 años, sobre todo a Luis y Felipe con quienes forje un buen equipo y amistad.

Decir gracias frente al eterno cariño y apoyo incondicional recibido.

Resumen

Los *CubeSats*, nanosatélites de bajo costo, se utilizan ampliamente en diversas aplicaciones espaciales, incluida la observación terrestre. En estas misiones, la orientación de la carga útil precisa hacia la Tierra es crucial, pero está limitada por el rendimiento y el costo de los componentes del *CubeSat*. Este estudio aborda el desafío de encontrar el *trade-off* ideal entre costo y rendimiento en el Subsistema de Determinación y Control de Actitud (ADCS) para misiones de observación terrestre.

Se implementó una suite de simulación para evaluar el rendimiento y el costo al utilizar diferentes componentes del ADCS. Los componentes fueron clasificados en tres niveles de calidad, y se cuantificaron los *Measures of Performance* (MoP) relacionados con la orientación, siendo estas la exactitud de apuntamiento, el *jitter*, la agilidad y el *drift*.

Para el desarrollo del simulador, la dinámica orbital se modeló utilizando SGP4, mientras que el ADCS se implementó mediante algoritmo TRIAD, un error representativo del filtro de Kalman extendido (EKF), el diseño del controlador PD y la utilización de modelos orbitales. Con ello, se obtuvo como resultado un control más ágil utilizando actuadores de alta calidad, y una mejor exactitud de apuntamiento dependiendo mayormente del tipo de sensor a utilizar. Se presentaron diferentes respuestas a la vibración mecánica del *jitter* que dependen netamente de una buena calidad del sensor, y una menor sensibilidad en la acción de control del actuador.

Los resultados mostraron un trade-off entre rendimiento y costo, donde una mayor calidad de componentes conllevó costos adicionales dentro de los límites definidos por los *System Engineering* (SE) *envelopes*, faltando considerar la implementación del EKF y el ruido en el giroscopio para completar el simulador, teniéndolo en consideración para trabajos futuros.

Palabras clave: *CubeSat*, ADCS, Capacidad de apuntamiento, Systems Engineering, Measures of Performance

Abstract

CubeSats, low-cost nanosatellites, are widely used in various space applications, including Earth observation. In these missions, precise orientation towards Earth is crucial, but it is limited by the performance and cost of CubeSat components. This study addresses the challenge of finding the ideal trade-off between cost and performance in the Attitude Determination and Control Subsystem (ADCS) for Earth observation missions.

A simulation suite was implemented to evaluate the performance and cost of using different ADCS components. Components were classified into three levels of quality, and Measures of Performance (MoP) related to orientation were quantified. These included pointing accuracy, which is the absolute error between the required and obtained axis, jitter represented as the power spectral density of the high-pass filter response, agility as the settling time with which the CubeSat stabilizes, and drift represented by the minimum angular velocity to prevent the satellite from leaving orbit.

For simulator development, orbital dynamics were modeled using SGP4, while ADCS was implemented using the TRIAD algorithm, an-Extended Kalman Filter (EKF) representative error, PD controller design, and orbital models. As a result, more agile control was achieved using high-quality actuators, and better pointing accuracy depended mostly on the type of sensor used. Different responses to mechanical vibration-induced jitter were presented, depending heavily on sensor quality, and less sensitivity in the actuator control action.

The results showed a trade-off between performance and cost, where higher component quality entailed additional costs within the limits defined by System Engineering (SE) envelopes. The implementation of the EKF and gyroscopic noise was not considered, leaving room for future work to complete the simulator.

Keywords: *CubeSat,* ADCS, Pointing capability, Systems Engineering, Measures of Performance

Tabla de Contenidos

Lista de Tablas

Lista de Figuras

Glosario

ACS	:	*Attitude Control Subsystem*
ADCS	:	*Attitude Determination and Control Subsystem*
COTS	:	*Commercial off the shelf*
EE	:	*Espacio Estado*
EDO	:	*Ecuaciones diferenciales ordinarias*
EKF	:	*Extended Kalman Filter*
F, f	:	*Matriz jacobiana de f, Dinámica no lineal del sistema*
GPS	:	*Global Position System*
H, h	:	*Matriz jacobiana de h, Medición no lineal del sistema*
I	:	*Momentos principales de inercia* $[kg \cdot m^2]$
L	:	*Momento angular* $\left[\frac{kg \cdot m^2}{s}\right]$
LEO	:	*Low Earth Orbit*
LVLH	:	*Local vertical Local horizontal (u orbital dentro del informe)*
M	:	*Triada*
MBSE	:	*Model-Based Systems Engineering*
MoP	:	*Measures of Performance*
PID	:	*Proporcional-Integrativo-Derivativo*
PSD	:	*Potential Spectral Density (Densidad espectro potencia)*
q	:	*Cuaternión*
r o $\vec{P}$, $\hat{r}$	:	*Posición del satélite, componente del vector de observación*
$\hat{s}$	:	*Componente del vector de referencia*
SE	:	*System Engineering*
SGP4	:	*Simplified General Perturbations 4*
SMAD	:	*Space Mission Analysis Design*
STK	:	*Systems Toolkit*
TLE	:	*Two Line Elements*
TRIAD	:	*Tri-axial Attitude Determination*
u, U	:	*Entrada de control, Unidad*
V, v, $\vec{V}$	:	*Vector de observación, ruido de la medición, vector velocidad*
W, w	:	*Vector de referencia, ruido del proceso del estado*
x	:	*Estado del sistema*
y	:	*Salidas del sistema*
z	:	*Medición del sistema*

Letras griegas

μ	:	Constante gravitacional de la Tierra $\left[\frac{km^3}{s^2}\right]$
ω	:	*Velocidad angular* $\left[\frac{rad}{s}\right]$
τ	:	*Torque [Nm]*
φ, θ, ψ	:	*Phi, theta y psi (ángulos de Euler)* [°]
Ω	:	*Right Ascension of the Ascending Node*

CAPÍTULO 1: Introducción

1.1 Contexto

En las últimas décadas, la revolución tecnológica ha posibilitado el desarrollo e implementación de satélites en órbita terrestre baja (LEO) a una escala sin precedente [1]. Entre estos, los *CubeSats* han surgido como una solución eficiente y versátil para una amplia gama de aplicaciones como lo son la observación terrestre y las comunicaciones. Los *CubeSats* son nanosatélites que se ajustan a un estándar que especifica sus dimensiones y diseño. Estos vienen en varios tamaños, siendo los más comunes 1U, 3U, 6U y 12U. La "U" en estas designaciones de tamaño significa "unidad" y se refiere al tamaño de un *CubeSat* en términos del número de unidades cúbicas de 10 x 10 x 11.35 [cm^3] que lo componen [2]. Se puede observar la relevancia de este tipo de satélites según la cantidad de lanzamientos que se han realizado a través de los años y de los confirmados a futuro en la Figura 1.

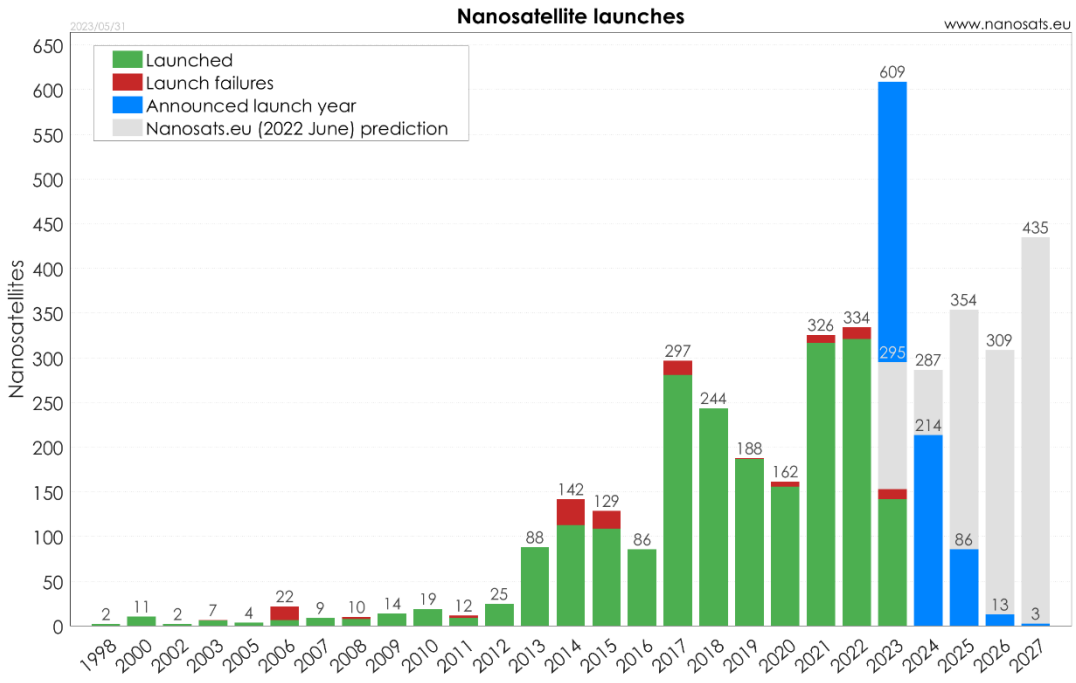


Figura 1. Cantidad de nanosatélites lanzados a través de los años [1].

Los *CubeSat* también presentan la ventaja de ser económicos respecto a los demás satélites grandes para funciones en LEO, como se demuestra en la Tabla 1. De las constelaciones de *CubeSat* con diferentes objetivos expuestas en la tabla, EPICHyper aun siendo las más cara de los *CubeSat*, sigue siendo más económica que los proyectos de satélite grandes de alta relevancia en LEO ya lanzados a través del tiempo.

Tabla 1. Costos de producción y lanzamiento de satélites de diferentes tamaños en LEO.

Constelación *CubeSat*	**Costo [USD]**	**Large/medium satellites**	**Costo [USD]**
AeroCube-14 [3]	4.1 millones USD	Starlink satellite [4]	15.25 millones USD
Deorbitsall [3]	3.2 millones USD	ISS satellite [5]	105 mil millones USD
Diamond 6U [3]	5.95 millones USD	Hubble Space Telescope [6]	1.5 mil millones USD
EPICHyper [3]	12.1 millones USD		

Dentro de estos nanosatélites, una aplicación con gran porcentaje de ocurrencia en el ámbito comercial está enfocado en la observación terrestre mediante la utilización de cargas útiles ópticas [3]. Esta aplicación consiste en la captura de imágenes de la superficie terrestre utilizando sensores que detectan distintas partes del espectro electromagnético. Estos sensores ópticos funcionan al detectar la luz reflejada (generalmente luz infrarroja o visible) en la superficie de la Tierra [7].

Para llevar a cabo este tipo de misión, se requiere de una capacidad de apuntamiento con el fin de cumplir el objetivo de capturar imágenes con la calidad y resolución requeridas. El concepto de apuntamiento se define como la capacidad del satélite para ajustar su actitud, logrando un cambio de orientación hacia un objetivo deseado [8]. Para lograr apuntar hacia la Tierra, se debe tener en cuenta los distintos marcos de referencia disponibles para representar los cambios de orientación, mostrándose en la Figura 2 los más importantes, como lo son la referencia en el cuerpo (body) y la referencia inercial externa.

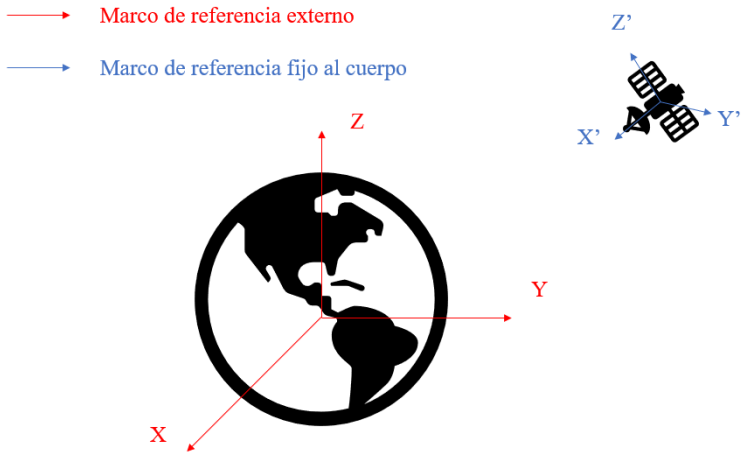


Figura 2. Marco de referencia fijo al cuerpo (satélite) y marco de referencia inercial (Tierra).

Para determinar/caracterizar la capacidad de apuntamiento, es necesario medir y evaluar su desempeño a través de los *Measures of Performanc*e (MoP) relevantes para las misiones de observación terrestre. En estudios anteriores [8, 9, 10, 11], se proporciona una definición cualitativa

de los siguientes MoP de apuntamiento, los cuales están condicionados por la calidad del hardware y software correspondientes al subsistema de determinación y control de actitud (ADCS) [8]:

- Exactitud de apuntamiento
- Estabilidad de la carga útil (o *drift* según [8])
- *Jitter*
- Agilidad

En conjunto con los MoP de apuntamiento, es necesario tener en consideración los *System Engineering* (SE) *envelopes* correspondientes al tipo de misión y satélite que se va a utilizar, los cuales son principalmente la potencia, la masa, el tamaño y el precio. Los SE *envelopes* son restricciones técnicas y operativas aplicables al *CubeSat* en fase de diseño, desarrollo y operación. Estos parámetros, una vez definidos de acuerdo con su relevancia, se basan principalmente en la elección de la carga útil, ya que es el punto de partida para iniciar el diseño y desarrollo de los demás subsistemas del satélite, en especial el ADCS que es crucial para el apuntamiento del satélite [8].

Dado que existen requerimientos establecidos para los SE *envelopes* del satélite y también para los MoP de apuntamiento dependiendo de la complejidad de la misión de observación terrestre, se debe encontrar un *trade-off* ideal entre ambos aspectos con el fin de satisfacer las necesidades de la misión. Es por ello por lo que los aspectos de la misión, tales como la dinámica orbital (parámetros orbitales y perturbaciones presentes en LEO) y la geometría del satélite, así como el modelamiento del ADCS se deben definir claramente con el objetivo de conocer el gasto correspondiente de la misión, así como su desempeño de apuntamiento. La Figura 3 presenta un diagrama que sintetiza el análisis reciente, ofreciendo una visualización que grafica tanto los parámetros de costo como los de rendimiento.

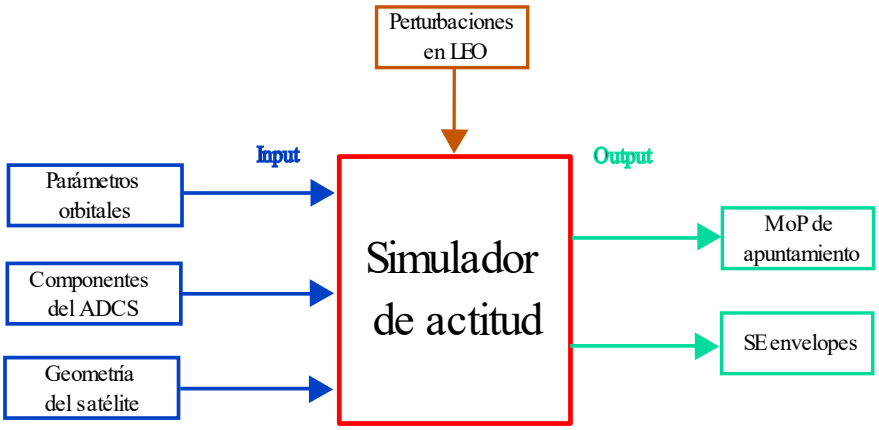


Figura 3. Representación gráfica de la relación entre aspectos de la misión/diseño ADCS respecto a costo y rendimiento.

Existen estudios en donde se simulan diferentes aspectos del ADCS como los componentes físicos [12], algoritmos de determinación de actitud [13] y controladores [14], en el cual analizan y comparan con que partes del ADCS consiguen mejores rendimientos, mostrando la potencia gastada como el costo más relevante. También existen estudios en los cuales realizan la simulación del ADCS completa para estudiar un *CubeSat*, especificando su diseño y sus resultados de rendimiento [15, 16, 17].

En [18] y [19] se presentan herramientas/simuladores capaces de implementar la dinámica orbital y de actitud, ofreciendo una interfaz gráfica de los movimientos traslacionales y rotacionales con sus respectivos modelos de perturbación, logrando cuantificar algunos MoP de apuntamiento y el costo en base a los SE *envelopes*. En [18] la herramienta tiene por nombre *Spacecraft Control Toolbox* y está dividida en tres secciones según los requerimientos de las misiones, con la capacidad de entregar resultados en base a la dinámica rotacional del satélite con su respectiva interfaz gráfica, incluyendo el consumo de energía para la versión económica, hasta recuperar los errores de apuntamiento y modelar sensores y actuadores en las versiones profesionales/académicas. Por otro lado, en [19] se muestran herramientas y plantillas disponibles en el *Aerospace Blockset* de MATLAB, en el cual se puede modelar un *CubeSat* según indicaciones específicas del satélite y de la órbita, además de simularlo mediante la inclusión de la herramienta *Simulink Animation 3D*. Incluso se pueden agregar herramientas de alta complejidad para obtener los SE e*nvelopes* en base a requisitos impuestos basados en la *Model-Based Systems Engineering* (MBSE).

Además, hay softwares como Valispace, que son capaces de únicamente analizar *budgets* de ingeniería, utilizados para conocer de manera genérica los gastos de los SE *envelopes* por cada componente del satélite y validar los requisitos impuestos para una misión, cuyo ejemplo en un uso común se presenta en [20]. Este tipo de software es capaz de entregar una visión general sobre análisis de satélites en cuanto a costos temporales, monetarios, de potencia, etc., al utilizar funciones y suma de requisitos hijos (de bajo nivel o más específicos) para cumplir el objetivo general.

Con esta información, se muestra que existen simuladores o herramientas de MATLAB capaces de entregar MoP de apuntamiento y el costo del hardware y software implementado para el ADCS del *CubeSat* (enfocándose principalmente en la potencia). Sin embargo, estas herramientas provienen de un lenguaje de programación/software que no es gratuito (alto costo monetario) y requieren de una curva de aprendizaje para su uso completo con el objetivo de llegar a analizar los resultados. Teniendo esto en cuenta, el simulador propuesto se vislumbra como una herramienta útil de análisis para el ADCS de un *CubeSat*, en el cual, al entregarle la información adecuada del satélite y la órbita, se obtiene el rendimiento y costo del apuntamiento, siendo capaz de aportar al desarrollo tecnológico de los *CubeSat* de manera simple y eficiente, con componentes actualizados en diferentes niveles y en un lenguaje de programación de libre acceso.

1.2 Condiciones de diseño

- Características de los *CubeSat*: Se deben considerar las restricciones y limitaciones típicas de los *CubeSats* en términos de los *SE envelopes* elegidos. Esto influirá en el diseño del simulador para que sea realista y práctico en el contexto de *CubeSat*
- Órbitas y perturbaciones en LEO: El simulador funcionará solo para orbitas de baja altura, por lo que solo se consideraran aquellas orbitas con parámetros orbitales a menos de 1000 [km] de altura. Mismo caso para las perturbaciones, que en LEO aquellas que tienen más relevancia son el arrastre atmosférico y la gravedad no esférica (J2).
- Selección de los componentes del ADCS: Se utilizarán actuadores, sensores y algoritmos típicos en *CubeSat*, siendo de tamaño reducido y de costo limitado según lo explicitado en los SE

envelopes. Además, se integrará dentro del proyecto aquellos componentes que representen el uso de la mayoría del tipo de misiones de estos satélites.

- Lenguaje para aplicar el simulador: El Departamento de Ingeniería Mecánica (DIM) cuenta con licencias para MATLAB, un software ampliamente utilizado en ingeniería y que es adecuado para crear simuladores. Sin embargo, se considera el uso de Python como una alternativa viable al ser un lenguaje de programación de libre acceso y gratuito. Se elige Python sobre MATLAB como una decisión de diseño debido a factores como la disponibilidad del simulador, capacidades especificas del lenguaje y necesidades del proyecto.

1.3 Objetivos

Objetivo General: Diseñar e implementar una arquitectura de simulación para evaluar *Measures of Performance* (MoP) de apuntamiento de *CubeSats* en misiones de observación terrestre ópticas para órbitas LEO en función de los SE *envelopes* correspondientes.

Los objetivos específicos del proyecto son:

- *OE1*: Caracterizar los componentes del ADCS por niveles según los SE *envelopes* para las misiones del tipo observación terrestre.
- OE2: Cuantificar los cuatro MoP de apuntamiento encontrados en trabajo previo.
- *OE3*: Recopilar información sobre la relación existente entre los componentes del ADCS con los MoP de apuntamiento.
- *OE4*: Investigar herramientas y modelos para la implementación de la dinámica orbital y de actitud en LEO para su uso en el simulador.
- *OE5*: Diseñar e implementar una arquitectura de simulación capaz de estimar el rendimiento y el costo de apuntamiento en base a entradas como el tipo de componentes del ADCS, la geometría y la órbita a analizar.
- *OE6*: Validar y obtener resultados de rendimiento y costo utilizando el simulador en base a información de al menos un *CubeSat.*

1.4 Metodología

Para el *OE1*, se utiliza el documento de la NASA "*State of the Art Small Space Technology*" [21] para obtener valores aproximados de costo de los sensores y actuadores que provienen de productos comerciales listos para usar (COTS) y que se utilizan en la actualidad en *CubeSats*.

Para cumplir el *OE2*, que es caracterizar cuantitativamente los MoP de apuntamiento descritos en trabajo previo [9], será necesario realizar un levantamiento de información en bibliografía con temática de diseño de satélite como el *Space Mission Analysis Design* (SMAD) [8], haciendo énfasis en el subsistema de determinación y control de actitud. También será necesario la búsqueda de artículos científicos que respalde la caracterización de estos MoP mediante la demostración del éxito utilizando los MoP de manera medible en los experimentos/misiones recopiladas.

Luego, en *OE3* se realizará una búsqueda bibliográfica tanto en artículos científicos como en el SMAD para identificar la correlación que existe entre los componentes del ADCS respecto a los MoP de apuntamiento descritos anteriormente.

Para el cumplimiento del objetivo *OE4*, se deben buscar herramienta adecuadas con el fin de obtener la dinámica orbital de satélite (lo que conlleva conocer los parámetros orbitales a través del tiempo y las perturbaciones presentes en LEO). Para diseñar los sensores, algoritmos y actuadores/controladores, se hará levantamiento de información a libros de diseño de ADCS o a autores que hayan publicado trabajo relacionado a este tópico, con el fin de encontrar la solución que mejor se adapte al simulador.

Posteriormente, se diseña e implementa una arquitectura de simulación propuesta en la *OE5*, se utilizará el lenguaje de programación Python para modelar la información ya obtenida sobre el ADCS y la dinámica orbital en el *OE4*, de manera tal que se tenga como base del código los modelos orbitales, los algoritmos y el control del satélite para terminar una primera iteración del simulador.

Finalmente, para validar el simulador en *OE6*, se utilizarán el *Two Line Elements* (TLE) de un *CubeSat* en órbita proveniente de Celestrak, con una fecha y tiempo de propagación determinada, incluyendo un estimado de la geometría y las condiciones iniciales correspondientes para observar los resultados tanto de rendimiento como de costo en base a los niveles impuestos dentro de la suite de simulación.

1.5 Carta Gantt

El proyecto sigue una estructura definida contenida en la carta Gantt mostrada en el Anexo A

CAPÍTULO 2: Marco Teórico

2.1 Sistemas de referencia

Para describir la dinámica orbital, de actitud y el diseño del ADCS, es necesario la definición de los marcos de referencia a utilizar. Su selección obedece a criterios y se describirán los utilizados para este trabajo a continuación.

- Sistema de referencia *body* o del cuerpo [22, 23]: La dinámica relativa de actitud se describe con respecto al marco de referencia del cuerpo en el *CubeSat*, desde la cual se realizan las mediciones, debido a que los sensores de actitud están fijados a su cuerpo. Para los marcos de referencia del cuerpo, el eje z apunta en la dirección del momento de inercia más alto, y los ejes x e y son paralelos a los vectores de área de las caras de la nave espacial, apuntando todos en las direcciones principales del satélite, como se observa gráficamente en la Figura 4.

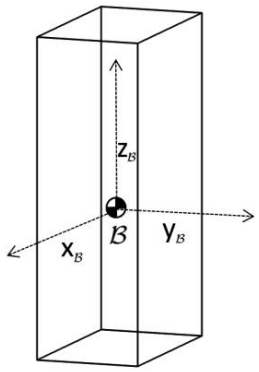


Figura 4. Marco de referencia del cuerpo [23].

- Sistema de referencia inercial [22, 23]: El Sistema de referencia inercial utilizado es el *Earth Centered Inertial* (ECI) debido a la necesidad de obtener vectores respecto a un marco de referencia no rotativo (asumiendo problema de dos cuerpos entre la Tierra y el satélite). El eje X apunta en la dirección del equinoccio de primavera. El plano XY es el plano ecuatorial de la Tierra, y el eje Z coincide con el eje de rotación de la Tierra y apunta hacia el norte. Este sistema de referencia se puede apreciar en la Figura 5.

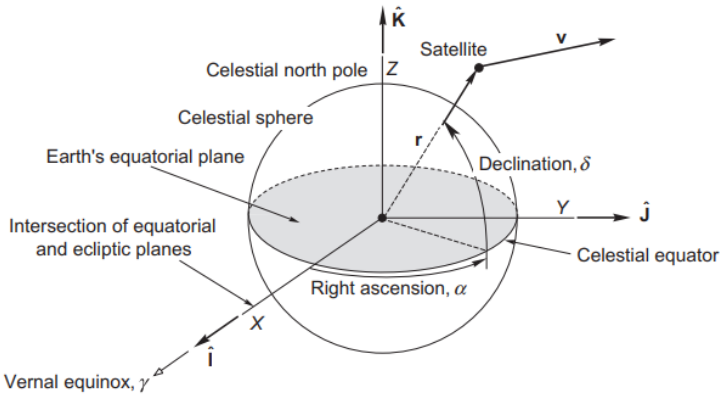


Figura 5. Marco de referencia inercial ECI [24].

- Sistema de referencia LVLH o Roll-Pitch-Yaw [25]: Se define un sistema coordenado que mantiene su orientación relativa a la Tierra a medida que la nave espacial se mueve en su órbita. Estas coordenadas son conocidas como roll, pitch y yaw (RPY), *Local Vertical-Local Horizontal* (LVLH) u "orbital" como también será llamada en este trabajo y se ilustra en la Figura 6. En este sistema, el eje yaw se dirige hacia el nadir (es decir, hacia el centro de la Tierra), el eje pitch se dirige hacia la normal negativa de la órbita, y el eje roll es perpendicular a los otros dos de manera que los vectores unitarios a lo largo de los tres ejes tienen la relación $\hat{R} = \hat{P} \times \hat{Y}$. Se utilizará este Sistema de referencia para notar la posición ideal de la carga útil.

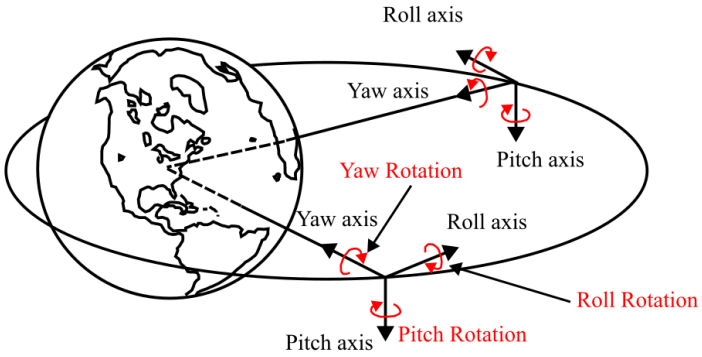


Figura 6. Marco de referencia RPY [25].

2.2 Dinámica orbital

Para el modelamiento de la suite de simulación se debe conocer el significado de los parámetros orbitales que se entregara como entrada, así como las ecuaciones que gobiernan el movimiento del satélite a través de la Tierra y las perturbaciones presentes a baja altura.

2.2.1 Parámetros orbitales

Si la masa de un satélite se considera insignificante en comparación con la masa de la Tierra, y bajo el supuesto de que la Tierra es esféricamente simétrica, la aceleración $\ddot{\boldsymbol{r}}$ de un satélite está dado por la ley de gravedad de Newton descrita a continuación:

$$\ddot{\boldsymbol{r}} = -\frac{GM_E}{r^3}\boldsymbol{r} \tag{1}$$

Donde $\boldsymbol{r}$ es el vector posición entre el satélite y la Tierra y GM_E es conocida como la constante de gravitacional de la Tierra y está dada por:

$$\mu = GM_E = 398600{,}4418 \left[\frac{km^3}{s^2}\right]$$

Al resolver la Ecuación (1), se obtiene la posición y la velocidad del del satélite respecto a la Tierra en cualquier instante de tiempo dependiendo del sistema de referencia a utilizar. Si bien se tiene una cuantificación del posicionamiento y el movimiento del satélite, generalmente se utiliza otra caracterización para definir la órbita, utilizando los elementos keplerianos, los cuales se presentan en la Figura 7.

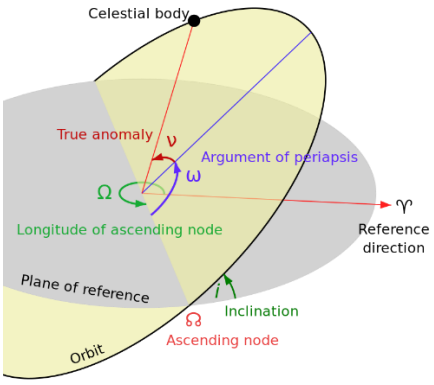


Figura 7. Elementos keplerianos [26].

Dichos elementos se definen brevemente a continuación, los que se pueden determinar desde la posición y la velocidad del satélite respecto a la Tierra mediante relaciones matemáticas obtenidas en [24].

- Excentricidad (e): Describe el alargamiento de la órbita. Si presenta valores entre 0 y 1 tendrá forma de elipse. Si es igual a 0 representa una órbita circular, mientras que si es igual a 1 tiene forma de parábola. Para casos mayores a 1 se presentan orbitas de trayectoria hiperbólica.
- Semieje mayor (a): Es la distancia entre el periapsis (distancia más cercana entre el satélite y la Tierra) y el apoapsis (distancia más lejana entre el satélite y la Tierra) dividido por 2. Representa el radio para orbitas circulares.
- Inclinación (i): Inclinación vertical de la elipse con respecto al plano de referencia (plano ecuatorial).
- *Right Ascension of the Ascending Node* (RAAN Ω): Es el ángulo medido entre la dirección I (dirección del equinoccio de primavera) y el nodo ascendente, o el punto donde un satélite cruza el ecuador moviéndose hacia arriba de sur a norte.
- Argumento del periapsis (ω): Se define la orientación de la elipse en el plano orbital, como un ángulo medido desde el nodo ascendente a la periapsis.
- Anomalía verdadera (ν): Define la posición del cuerpo orbitante a lo largo de la elipse en un tiempo específico.

2.2.2 Perturbaciones presentes en LEO

Existen perturbaciones en el espacio que afectan a los satélites en órbita, de las cuales algunas tienen más relevancia a bajas altura respecto de la Tierra. Estas se muestran a continuación:

Gravedad no esférica [27]: La Tierra no es una esfera perfecta y la masa se distribuye de manera no uniforme. A diferencia de las simplificaciones que se aplican a menudo en órbitas altas, donde la influencia de la Tierra se aproxima a una esfera, en LEO la distribución irregular de la masa terrestre y las variaciones en la altitud pueden generar perturbaciones significativas en las trayectorias de los satélites. Por lo tanto, como la fuerza de gravedad depende directamente de la masa, el campo gravitatorio reflejará esta falta de uniformidad.

Para lograr modelar la gravedad no esférica se utiliza una expansión armónica esférica, con modelos como el geopotencial que descompone el campo gravitatorio terrestre en una serie de términos, cada uno correspondiente a una armonía esférica y su respectiva magnitud. Dentro de los coeficientes utilizados dentro del modelo recién mencionado están los "J", siendo el J2 el principal para modelar el achatamiento de la Tierra. Otros J como el J3, J4, etc., modelan a mayor detalle la distribución másica de la Tierra.

Efectos atmosféricos [28]: Los efectos del arrastre y el oxígeno atómico (O) tienen implicancias para los satélites de baja altura (menor a 600 km). El arrastre se define como una fuerza resistiva que actúa sobre un objeto en movimiento a través de un fluido y tiende a disminuir su velocidad, cuyas implicancias son que acorta la vida útil del satélite. El arrastre depende de la densidad, la velocidad y también variará según cómo cambie la atmósfera (se expanda o se contraiga) debido a la variación en la actividad sola.

Por otro lado, debido a que en la atmosfera superior existe una mayor radiación, esto hace que se disocien los átomos de O_2 a O, los cuales son muy reactivos y potencialmente dañinos, degradando las superficies del *CubeSat* e interfiriendo con los sensores para la determinación de actitud.

2.3 Cinemática y dinámica de actitud

Para la cinemática y dinámica de actitud, el satélite ya no se asume como una partícula perturbada (como en el caso de la dinámica orbital), sino como un cuerpo rígido con masa. Con esto aparecen conceptos que serán definidos en esta sección.

2.3.1 Actitud de un satélite y sus representaciones

La actitud de un satélite se refiere a la orientación o posición que mantiene en el espacio mientras órbita alrededor de la Tierra u otro cuerpo celeste. Para describir esta actitud, se utilizan diversas representaciones matemáticas que permiten definir de manera precisa su orientación en el marco de referencia del cuerpo respecto al inercial, las cuales se presentan a continuación [25]:

- *Direction Cosine Matrix (DCM)*: Esta parametrización utiliza una matriz 3x3 para representar la orientación del satélite en relación con un sistema de referencia fijo. La matriz contiene nueve elementos que son los cosenos directores de los ejes del satélite en relación con los ejes de referencia. Es una representación matemáticamente precisa pero no es tan intuitiva como otras. Dicha representación se visualiza en la Figura 8.

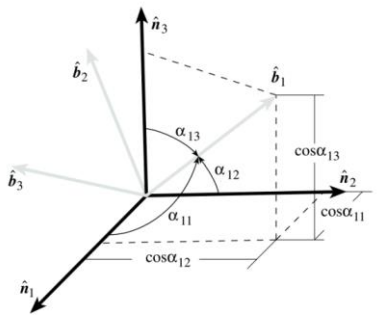


Figura 8. Representación gráfica de la primera fila de la matriz de cosenos directores [29]

- Euler axis/angle: En esta parametrización, se utiliza un vector tridimensional (el eje de Euler) junto con un ángulo para describir la orientación. El vector de Euler define el eje de rotación, mientras que el ángulo especifica la magnitud de la rotación alrededor de ese eje, como se muestra en la Figura 9. Es útil para representar giros simples y es intuitiva.

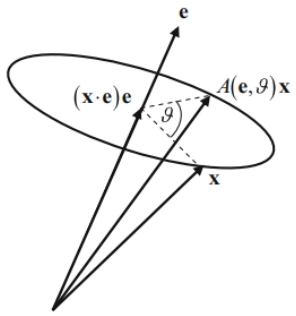


Figura 9. Representación de un cambio de orientación en Euler axis/angle de x [27].

- Euler angles: Esta parametrización describe la orientación mediante tres ángulos, generalmente llamados phi (φ), theta (θ) y psi (ψ), que representan las rotaciones en torno a los ejes específicos (por ejemplo, X, Y y Z). Se presenta en la Figura 10 las rotaciones realizadas por esta parametrización.

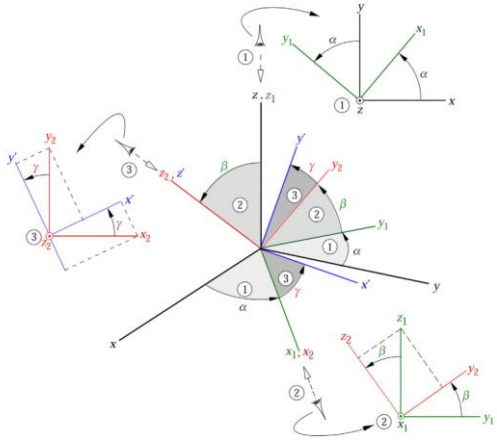


Figura 10. Secuencia clásica de Euler de tres rotaciones que transforman xyz en x'y'z' [24].

- Cuaternión: Esta parametrización utiliza cuatro parámetros para representar la orientación, siendo tres de estas vectoriales y una escalar. Se discutirá más a fondo en la siguiente sección.

2.3.2 Cuaterniones y cinemática de cuaterniones

En este trabajo, la parametrización seleccionada para la descripción de la actitud es el cuaternión. Los cuaterniones tienen múltiples ventajas en comparación con otras parametrizaciones de actitud. Por ejemplo, en la parametrización de ángulos de Euler, la propagación de la actitud no es suave. Sin embargo, esta suavidad es fundamental para el correcto funcionamiento de métodos de estimación como el Filtro de Kalman [25].

Por otro lado, la desventaja de la parametrización de la matriz de cosenos directores es que conduce a una descripción de la actitud utilizando nueve elementos no independientes, y cumple con seis restricciones impuestas por la ortogonalidad de la matriz de actitud que son redundantes.

La cantidad mínima de elementos que se pueden utilizar para describir la actitud sin singularidades son cuatro. A partir del hecho de que cualquier rotación puede describirse utilizando un solo eje de rotación y un ángulo que describe la rotación alrededor de este eje, el cuaternión se define mediante 4 elementos, con una parte vectorial y una parte escalar como se muestra a continuación [24]:

$$\hat{\boldsymbol{q}} = \begin{Bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{q} \\ q_3 \end{Bmatrix} = \begin{Bmatrix} sin\left(\frac{\theta}{2}\right)\hat{\boldsymbol{u}} \\ cos\left(\frac{\theta}{2}\right) \end{Bmatrix}$$

La expresión **q** es la parte vectorial ($\mathbf{q} = q_0\hat{\boldsymbol{\imath}} + q_1\hat{\boldsymbol{\jmath}} + q_2\hat{\boldsymbol{k}}$) y q_3 es la parte escalar. La representación mostrada representa un cuaternión pasivo, el cual se utiliza para rotar el sistema de coordenadas en sí (sin rotar el vector). Si se quiere rotar el vector sin el sistema coordenado, se utiliza un cuaternión activo, y se obtiene cambiando la componente escalar al inicio, tal y como se representa en la siguiente expresión:

$$\hat{\boldsymbol{q}} = \begin{Bmatrix} q_3 \\ q_0 \\ q_1 \\ q_2 \end{Bmatrix} = \begin{Bmatrix} q_3 \\ \boldsymbol{q} \end{Bmatrix}$$

Sabiendo esto, se puede definir la cinemática de la actitud del satélite utilizando cuaterniones mediante la Ecuación (2), sabiendo que ω_0, ω_1 y ω_2 son las velocidades angulares en el marco de referencia cuerpo del satélite:

$$\frac{d}{dt}\hat{\boldsymbol{q}} = \frac{1}{2}\begin{bmatrix} 0 & \omega_2 & -\omega_1 & \omega_0 \\ -\omega_2 & 0 & \omega_0 & \omega_1 \\ \omega_0 & -\omega_1 & 0 & \omega_2 \\ -\omega_0 & -\omega_1 & -\omega_2 & 0 \end{bmatrix}\hat{\boldsymbol{q}} \tag{2}$$

Además, se tiene la multiplicación de cuaterniones, la cual será útil para representar las rotaciones entre los sistemas de referencia y aproximaciones discretas que utilizan esta operación, la cual se representa en la Ecuación (3):

$$\hat{\boldsymbol{q}} \cdot \hat{\boldsymbol{r}} = \begin{bmatrix} q_3 r_0 + q_0 r_3 + q_1 r_2 - q_2 r_1 \\ q_3 r_1 + q_1 r_3 + q_2 r_0 - q_0 r_2 \\ q_3 r_2 + q_2 r_3 + q_0 r_1 - q_1 r_0 \\ q_3 r_3 - q_0 r_0 - q_1 r_1 - q_2 r_2 \end{bmatrix} \tag{3}$$

Si bien esta parametrización no tiene una representación física obvia, se puede describir la rotación del satélite a través del tiempo mediante un integrador numérico, sabiendo la condición inicial tanto del cuaternión como de la velocidad angular.

2.3.3 Dinámica de actitud

La ecuación que describe la variación del vector momento angular a través del tiempo para un torque aplicado en un marco de referencia del cuerpo representa la dinámica de actitud y se muestra en la Ecuación (4) [24]:

$$\boldsymbol{I}\frac{d\boldsymbol{\omega}}{dt} = -\boldsymbol{\omega} \times \boldsymbol{I}\boldsymbol{\omega} + \boldsymbol{\tau} \tag{4}$$

Sabiendo que ω es el vector de velocidad angular instantánea en el cuerpo, **I** son los momentos principales de inercia y τ son los torques aplicados. Esta ecuación se puede representar también según sus componentes en i, j y k:

$$\dot{\omega_0} = \frac{\omega_1\omega_2(I_y - I_z)}{I_x} + \frac{\tau_x}{I_x}$$
$$\dot{\omega_1} = \frac{\omega_0\omega_2(I_x - I_z)}{I_y} + \frac{\tau_y}{I_y}$$
$$\dot{\omega_2} = \frac{\omega_0\omega_1(I_x - I_y)}{I_z} + \frac{\tau_z}{I_z}$$

Los torques externos son los provocados por los actuadores y por perturbaciones externas en orbitas de baja altura. Las ultimas mencionadas generalmente se simplifican a relaciones para el peor de los casos analizados. Estas perturbaciones que afectan a la dinámica rotacional del satélite son cuatro y se discuten a fondo en el Anexo B.

2.4 Subsistema de determinación y control de actitud

El subsistema de determinación y control de actitud de un *CubeSat* es el responsable de determinar su orientación en el espacio y controlarla respecto a un objetivo durante un periodo específico. También se requiere para sobrevivir en el entorno espacial, controlando el satélite en orientaciones tales que se genere energía apuntando las celdas fotovoltaicas hacia el sol.

El proceso para poder describir el movimiento del satélite se describe en tres pasos o procesos los cuales son llamados *Guidance, Navigation and Control* (GNC). Cada uno se describe según los componentes del ADCS, como se muestra a continuación [30]:

- *Navigation*: Es el primer paso por seguir, y determina la actitud del satélite y la tasa de rotación mediante los sensores tanto inerciales como de medición externa. Responde a la pregunta ¿Dónde está el satélite?
- *Guidance*: Al determinar la actitud con los sensores, se utilizan algoritmos de determinación de actitud para estimar la orientación tanto inicial como a través del tiempo, para posteriormente utilizar algoritmos de determinación de actitud para estimar la orientación del satélite. Responde a la pregunta ¿Hacia dónde quiere ir el satélite?
- *Control*: Ya conocido hacia donde quiere ir el satélite, se genera el cambio de actitud mediante la implementación de torques utilizando controladores en conjunto con actuadores. Responde a la pregunta ¿Como dirijo el satélite hacia allá?

2.4.1 Navigation (Sensores) [9]

Para la determinación de actitud se utilizan componentes físicos llamados sensores, los cuales miden su orientación en base a la inercia del satélite como también observando las estrellas circundantes/cuerpos celestes o midiendo fuerzas representativas de una posición en particular. Los sensores comúnmente utilizados en *CubeSat* se presentan en la Tabla 2, en conjunto con su descripción.

2.4.2 Guidance (Algoritmos) [15]

En esta sección se proporciona una descripción de los diferentes métodos de determinación de actitud que se consideran para el simulador. En primera instancia, existen dos tipos principales de métodos de determinación de actitud. El primero de ellos es el método determinístico que utiliza la información de las lecturas de los sensores a lo largo de la misión y las comparan con modelos informáticos para calcular la actitud actual. Por otro lado, existen los estimadores recursivos que procesan las lecturas de sensores actuales y las compara con la última estimación de actitud para crear una nueva estimación.

Enfoque determinista de la determinación de actitud

Para la determinación inicial de la actitud, se requiere un enfoque determinista. Existen varios algoritmos deterministas diferentes para la determinación de la actitud, describiendo algunos a continuación:

TRIAD method: La solución Tri-axial Attitude Determination (TRIAD) requiere dos conjuntos de vectores: un vector de observación de cada uno de los dos sensores ubicados en el satélite (V_1 y V_2), y un vector de referencia para cada observación en términos de su dirección de referencia inercial (W_1 y W_2). Con estos vectores se crean triadas de referencia (M_{ref}) y de observación (M_{obs}) descritas en la Tabla 3.

Tabla 2. Sensores utilizados en *CubeSat* [9].

Sensores	Descripción
Giroscopios	Los giroscopios miden la tasa de cambio de la orientación angular respecto al marco inercial del satélite [°/s]. Los giróscopos proporcionan información sobre los movimientos de rotación en los tres ejes (roll, pitch y yaw)
Sensores de sol	Los sensores solares se utilizan para estimar la dirección del Sol en el marco de referencia del cuerpo del satélite.
Magnetómetros	Los magnetómetros determinan el campo magnético de la Tierra, midiendo su dirección y su fuerza en [nT]
GPS	Mediante el receptor GPS se obtiene la posición tridimensional (latitud, longitud y altitud) con alta precisión. Esto permite al *CubeSat* conocer su ubicación en la órbita terrestre.
Star Tracker	Un Star Tracker o contador de estrellas es un sistema de sensores cuya función principal es determinar con exactitud la actitud del *CubeSat* utilizando las estrellas circundantes como referencia.

Tabla 3. Descripción de las triadas de referencia y de observación.

Triada	Componentes de la triada
$M_{obs} = [\widehat{r_1}\ \widehat{r_2}\ \widehat{r_3}\ ]$	$\widehat{r_1} = \widehat{V_1}\ ;\ \widehat{r_2} = \frac{\widehat{V_1} \times \widehat{V_2}}{\lvert\widehat{V_1} \times \widehat{V_2}\rvert}\ ;\ \widehat{r_3} = \widehat{r_1} \times \widehat{r_2}$
$M_{ref} = [\widehat{s_1}\ \widehat{s_2}\ \widehat{s_3}\ ]$	$\widehat{s_1} = \widehat{W_1}\ ;\ \widehat{s_2} = \frac{\widehat{W_1} \times \widehat{W_2}}{\lvert\widehat{W_1} \times \widehat{W_2}\rvert}\ ;\ \widehat{s_3} = \widehat{s_1} \times \widehat{s_2}$

Una vez que se calculan las tríadas de observación y referencia, se puede encontrar la solución TRIAD. Esta solución es la matriz de cosenos directores A que se define de la siguiente manera:

$$A = M_{obs}\left(M_{ref}\right)^T$$

Esta matriz representa la rotación desde el marco de referencia del cuerpo del satélite al marco de referencia inercial fijo a la Tierra. Una vez que se conoce esta matriz, la actitud del satélite se puede expresar en términos del marco de referencia inercial fijo a la Tierra. Cuando se utiliza el método TRIAD, el sensor más exacto debe elegirse siempre como V_1 con su correspondiente vector de referencia W_1.

Por otro lado, existen algoritmos que ofrecen un mínimo error sin la necesidad de elegir el sensor más exacto como el V_1. Para ello, buscan resolver la ecuación de Wahba, la cual minimiza el error de determinación de actitud al asignarle pesos a los vectores de referencia y de observación. Estos métodos son el q-method y el QUEST [15], los cuales no se utilizarán durante este trabajo, ya que como se muestra en Vélez [15], q-method presenta mayores errores en la determinación de actitud respecto a los otros algoritmos, mientras que QUEST en las mismas simulaciones tiene una calidad similar al TRIAD, con mayor costo computacional y una mayor dificultad de implementación.

Enfoque recursivo de la estimación de actitud

Aunque se necesita un método determinista para la adquisición inicial de la actitud, un método recursivo a menudo es más eficiente para el mantenimiento del conocimiento de la actitud. A diferencia de los métodos deterministas, los métodos recursivos utilizan solo la lectura actual del sensor para calcular el error de actitud a partir de la estimación anterior. El método recursivo más común es un Filtro de Kalman.

Los Filtros de Kalman se utilizan para estimar los estados futuros de un sistema dinámico lineal afectado por ruido. El algoritmo consta de dos fases: la fase de predicción y la fase de actualización. Durante la fase de predicción, el filtro utiliza ecuaciones dinámicas del sistema preprogramadas para calcular la estimación a priori de la nueva actitud del satélite. Durante la fase de actualización, el filtro utiliza las lecturas actuales del sensor para determinar la estimación a posteriori de la actitud actual del satélite y calcular el error de estimación a partir de la predicción de la actitud.

Dado que el Filtro de Kalman tiene un proceso de dos pasos, utiliza un paso de tiempo discreto, k. La naturaleza discreta del Filtro de Kalman funciona bien para el *CubeSat*, ya que el paso de tiempo k puede configurarse fácilmente como el intervalo de tiempo entre las mediciones de los sensores. Sin embargo, el sistema dinámico del satélite al no ser lineal (dependiente del tiempo) se necesita utilizar

el Filtro de Kalman Extendido (EKF). La naturaleza discreta del EKF compensa la dependencia del tiempo en el modelo del satélite. Las demás no linealidades en las ecuaciones dinámicas se abordan a través de matrices Jacobianas, que están compuestas por las derivadas parciales de primer orden de las ecuaciones dinámicas del sistema. Estas matrices Jacobianas permiten al EKF linealizar el sistema no lineal en la estimación actual y se deben calcular nuevas matrices para cada paso de tiempo.

El modelo dinámico no lineal y las mediciones se definen de la siguiente manera:

$$x_{k+1} = f_k(x_k, u_k) + w_k$$
$$z_k = h_k(x_k) + v_k$$

Donde x_{k+1} es el estado actual del sistema, x_k es el estado previo del sistema, u_k es la entrada de control, z_k es la medición del sistema, f_k representa la dinámica no lineal del sistema y h_k representa la medición no lineal. El ruido del proceso del estado se representa como w_k y el ruido esperado de la medición es v_k. Las matrices Jacobianas F y H de f_k y h_k respectivamente se pueden encontrar tomando la derivada parcial de f y h con respecto a x.

$$F(\hat{x}, t) = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}}$$
$$H(\hat{x}, t) = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}}$$

La estimación a priori y la matriz de error de covarianza pueden ser obtenidas por las siguientes ecuaciones:

$$\hat{x}_k^- = f_k(\hat{x}_k)k$$
$$P_k^- = F_k P_{k-1} F_k^T + Q_k$$

La matriz Q_k representa la matriz de covarianza del ruido del modelo w_k. La ganancia de Kalman se calcula con la ecuación a continuación.

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R_k)^{-1}$$

La matriz R_k representa la matriz de covarianza del ruido del sensor w_k.

El siguiente paso es la actualización de la medición. Las ecuaciones para la estimación a posteriori del estado y la matriz de covarianza del error se encuentran a continuación.

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - h_k(\hat{x}_k^-))$$
$$P_k = (I - K_k H_k) P_k^-$$

2.4.3 Control (Controladores y actuadores)

Para el control del satélite, se requiere la orden del controlador para generar un torque mediante los actuadores.

Controladores

Controlador Proporcional-Derivativo (PD) y Proporcional-Integrativo-Derivativo (PID) [25]: Es un tipo de controlador ampliamente utilizado en automatización y control de procesos para regular

sistemas dinámicos y mantener una variable de proceso en un valor deseado o *setpoint*. A continuación, se explicará cada uno de los componentes y cómo funcionan juntos para controlar el sistema:

- Componente Proporcional (P): El término proporcional es la parte principal del controlador PID. Su función es proporcionar una respuesta inmediata a las desviaciones actuales entre la variable controlada y el valor deseado (error). La salida del término proporcional (P) es directamente proporcional al error actual, por lo que cuanto mayor sea el error, mayor será la corrección aplicada.
- Componente integrativo (I): El término integral es responsable de acumular el error a lo largo del tiempo y compensar errores persistentes o a largo plazo. El término integral responde a la acumulación de errores pasados, por lo que tiende a eliminar errores persistentes o sistemáticos. Esta componente ayuda a reducir el error constante (offset) y garantiza que el sistema alcance el setpoint. Sin el componente integral, el controlador podría quedarse con un error constante incluso si el controlador proporcional es capaz de mantenerlo bajo control.
- Componente derivativo (D): El término derivativo es sensible a la tasa de cambio del error. Se encarga de prevenir oscilaciones y estabilizar el sistema. La acción derivativa es capaz de prever la dirección en la que el error se está moviendo y disminuir la velocidad a la que se acerca al setpoint. Ayuda a suavizar las respuestas del sistema y evita que el controlador reaccione de manera brusca ante cambios repentinos en el error.

La salida del controlador se representa mediante la siguiente ecuación:

$$U = k_p P + k_i I + k_d D$$

Siendo k_p, k_i y k_d constantes de ajuste de ganancias proporcional, integral y derivativo respectivamente, determinando la magnitud de la contribución de cada termino de control en general.

Controlador Linear Quadratic Regulator (LQR) [31]: Este es un método de control óptimo utilizado en sistemas dinámicos lineales y de tiempo continuo. Su objetivo es encontrar la ley de control lineal que minimiza una función de costo cuadrática, teniendo en cuenta tanto el estado del sistema como la entrada de control. Para su uso se debe modelar el sistema según como se presenta a continuación, en donde x es el vector de estado, u es el vector de entrada de control, A es la matriz de estado y B es la matriz de entrada:

$$\dot{x} = Ax + Bu$$

Posterior a esto, se requiere el uso de una función costo cuadrática que se debe minimizar. La función de costo típicamente incluye términos que penalizan el error del estado y el esfuerzo de control, ponderados por matrices de ponderación Q y R, respectivamente, la cual se expresa como:

$$J = \int_0^\infty (x^T Q x + u^T R u) dt$$

El objetivo es encontrar el $u = -Kx$ que minimiza la función de costo, siendo K la ganancia del controlador. Esta matriz K se obtiene la resolver la ecuación de Riccati expuesta a continuación, donde P es la matriz simétrica definida positiva asociada con la solución de la ecuación de Riccati:

$$A^T P + PA - PBR^{-1}B^T P + Q = 0$$

Ya encontrada la matriz P, la ley de control óptimo se obtiene como $u = -Kx$ con $K = R^{-1}B^T P$. Esta ley se implementa en el sistema dinámico para estabilizarlo y minimizar la función de costo a lo largo del tiempo. El controlador LQR es particularmente eficaz para sistemas lineales y proporciona un enfoque sistemático para el diseño de controladores óptimos

Actuadores

Los actuadores son los que generan el torque necesario para el control del satélite. Los actuadores comúnmente utilizados en *CubeSat* se muestra en la Tabla 4, en conjunto con una breve descripción y un ejemplo utilizado:

Tabla 4. Actuadores utilizados en *CubeSat* [9].

Actuadores	Descripción
Magnetorquer	Los magnetorquers son dispositivos de control de actitud construidos utilizando bobinas electromagnéticas, que generan un torque a través de interacciones entre el campo magnético ambiental y dipolos magnéticos generados por este actuador.
Rueda de reacción	Una rueda de reacción es un motor acoplado a un disco de alta inercia que gira a gran velocidad a lo largo de un eje fijo del satélite. Este funciona aplicando un torque T en el disco, provocando un aumento en su momento angular h. Por conservación de momento angular (al haber ausencia de fuerzas externas) se genera un torque de igual magnitud, pero en sentido contrario que es aplicado en el *CubeSat*

2.5 System Engineering Envelopes

Los *Systems Engineering Envelopes* son restricciones técnicas y operativas aplicables al *CubeSat* en fase de diseño, desarrollo y operación. Las decisiones de diseño se basan en estos parámetros:

- Precio: ¿Cuál es el costo monetario de utilizar una u otra alternativa?
- Potencia: ¿Cuanta energía consume la alternativa a utilizar?
- Masa: ¿Cuánta masa se utiliza con la alternativa elegida respecto al total requerido?
- Tamaño: ¿Cuánto volumen ocupa el componente a utilizar?

En el contexto de este trabajo, se buscará cuantificar el costo respecto a precio, potencia, masa y tamaño, al utilizar distintos tipos de componentes del ADCS. Con esto se verá si en una misión de observación terrestre, cuál será el costo con el que se apuntó la carga útil hacia un lugar en específico de la Tierra.

2.6 Controlabilidad y observabilidad de un sistema de control

Un sistema dinámico se considera controlable si se pueden aplicar señales de control que accionen cualquier estado del sistema dentro de una cantidad de tiempo finita. Esta característica también se denomina accesibilidad. Por otro lado, se considera observable si todos sus estados pueden conocerse a partir de la salida del sistema.

Si se tiene un modelo de espacio de estados de tiempo continuo con N_x estados, N_y salidas y N_u entradas como se muestra a continuación:

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

Donde:

x : Estados del sistema
u : Entradas del sistema
y : Salidas del sistema

A, B, C y D: las matrices de espacio de estados con tamaños $N_x \times N_x$, $N_x \times N_u$, $N_y \times N_x$ y $N_y \times N_u$ respectivamente de valores reales o complejos.

El sistema es controlable y observable si la matriz de controlabilidad generada $Co = [B, AB, A^2B, \dots, A^{n-1}B]$ y la matriz de observabilidad $Ob = [C, CA, CA^2, \dots, CA^{n-1}]$ tienen un rango total, es decir, el rango es igual al número de estados del modelo de espacio de estados.

Es relevante tener en cuenta estos conceptos, ya que, para utilizar algunos algoritmos de estimación de actitud, es necesario que el sistema sea observable, como es el caso del EKF. Mismo caso para el uso de controladores, que dependen de si el sistema logra ser controlable para accionar el torque necesario para el sistema satelital

CAPÍTULO 3: Estado del Arte

En este capítulo se muestran herramientas que logran simular la dinámica orbital y la dinámica de actitud del satélite, además de visualizarlo mediante una interfaz gráfica. Algunos de ellos son simuladores que entregan alguno de los MoP de apuntamiento, así como también softwares especializados en el análisis de *budgets* de ingeniería, en los cuales se encuentran los SE *envelopes* descritos.

3.1 Spacecraft Control Toolbox [18]

Spacecraft Control Toolbox (SCT) para MATLAB le permite diseñar, analizar y simular naves espaciales. Este producto es utilizado en todo el mundo por organizaciones líderes en investigación y desarrollo y fabricantes de naves espaciales. Se proporcionan más de dos mil funciones para dinámica, simulación, análisis y diseño de actitud y órbita. Puedes construir un satélite utilizando las herramientas gráficas CAD; diseñar y analizar los sistemas de control; realizar análisis de perturbaciones y pruebe el sistema de control en una simulación de seis grados de libertad, todo en el lenguaje de programación MATLAB.

Existen tres módulos disponibles, los cuales se muestran a continuación:

- *CubeSat Toolbox*: Es el producto más básico disponible, a un costo de 495 dólares. Está diseñado para *CubeSat* específicamente y sus principales características son el modelamiento de la dinámica y control de actitud (perturbaciones, controladores PID en conjunto con actuadores), propagación de orbita, cambios de marco de referencia, visualización (2D y 3D) y planeamiento de la misión. En la Figura 11, se presenta un resumen de los datos obtenidos por el simulador una vez creado el diseño 3D en CAD. Por otro lado, en la Figura 12, se muestra en específico la ventana de control del simulador, mostrando tanto los cuaterniones de rotación, como los torques y las razones de cambio de la rueda de reacción montada en el *CubeSat.*

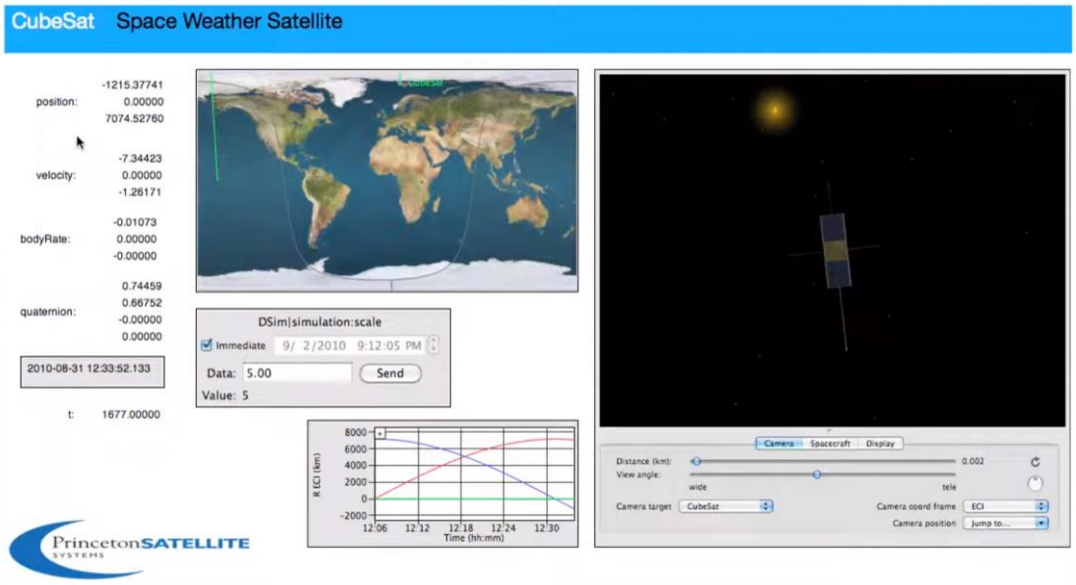


Figura 11. Resumen de la dinámica orbital y de actitud del satélite en *CubeSat Toolbox*.

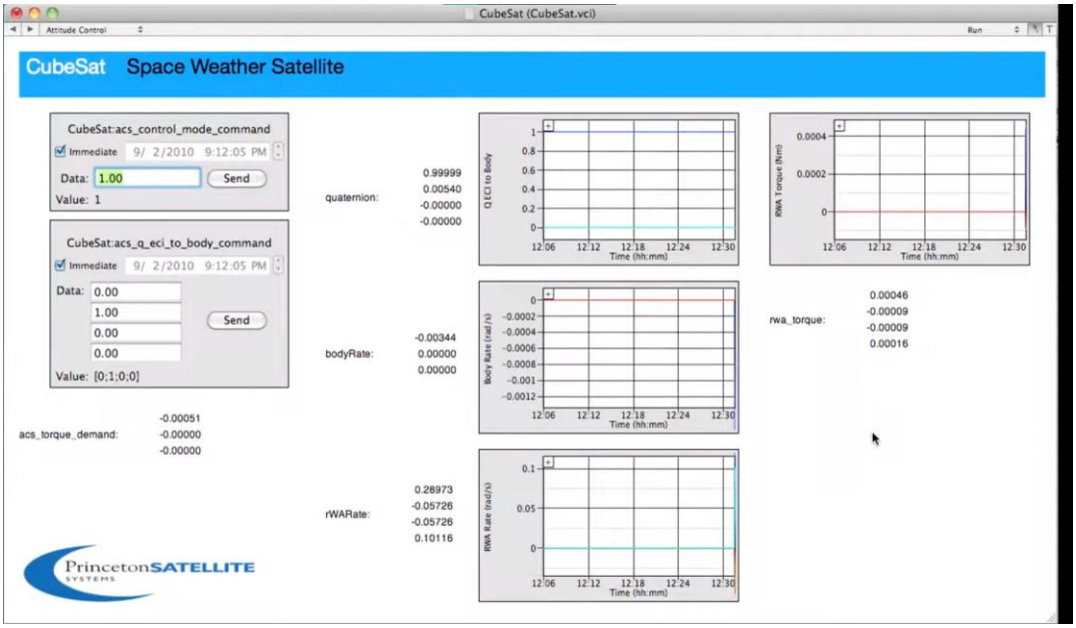


Figura 12. Ventana de control del simulador de un *CubeSat* en *CubeSat* Toolbox.

- *Spacecraft Control Toolbox* versión académica: La edición académica de la caja de herramientas contiene gran parte del software profesional *Spacecraft Control Toolbox*, incluidas las herramientas CAD, las funciones de diseño de control y dinámica de actitud, la mecánica orbital y el módulo *CubeSat*. Además, incluye esquemas de apuntamiento según marcos de referencia LVLH, sun-nadir *pointing* y latitude/longitude, y también otorga los *pointing budgets* más relevantes como la exactitud de apuntamiento. Esta versión cuesta 3995 dólares su adquisición y presenta un breve tutorial en el canal de YouTube de Princeton.
- *Spacecraft Control Toolbox* versión profesional: La edición profesional de *Spacecraft Control Toolbox* ofrece todo lo que hay en las versiones *CubeSat* y *Academic*, con la adición de herramientas avanzadas para la estimación de actitud y órbita, modelado de sensores y actuadores y análisis de subsistemas. Las aplicaciones de la caja de herramientas incluyen diseño de sistemas de control, simulación no lineal, análisis de órbitas y planificación de misiones, incluidas trayectorias interplanetarias, diseño y disposición de naves espaciales, estudios comerciales y visualización de actitudes y órbitas. La versión profesional cuesta 11995 dólares, siendo esta la más completa y a la vez más cara entre las tres mencionadas.

3.2 Ansys Systems Tool Kit (STK) [32]

STK es una plataforma de software líder en el mercado diseñada para el modelado y análisis de sistemas complejos y sus interacciones en una variedad de dominios, incluyendo espacio, defensa y aplicaciones aeroespaciales. Esta herramienta proporciona una serie de capacidades avanzadas que permiten a los ingenieros, científicos y analistas modelar, simular y visualizar sistemas dinámicos de manera efectiva.

Es relevante ya que es capaz de modelar un satélite mediante su diseño 3D, mostrando gráficamente como este se propaga a través de la órbita. También es capaz de simular los cambios de actitud del

satélite a través del tiempo y mostrar si efectivamente se está apuntando hacia la zona requerida de la tierra (exactitud de apuntamiento). En la Figura 13 se aprecia una imagen referencial de la simulación de un satélite en 3D y sus marcos de referencia cuerpo y LVLH. El costo de la licencia investigativa de este software es de alrededor de 4527 dólares por un año, que sería la versión útil para la realización del proyecto.

Figura 13. Imagen referencial de STK para simulación de un satélite.

3.3 *Aerospace Blockset* [19]

Dentro de este módulo en MATLAB, existen librerías capaces de modelar, simular y analizar *CubeSats* con facilidad. Algunas de las capacidades clave incluyen:

- Modelado de *CubeSats*: El modelo de plantilla es un ejemplo listo para simular que contiene un bloque de vehículo *CubeSat*. MATLAB permite a los usuarios modelar *CubeSat* para proporcionar una opción de planificación de misión de alto nivel/creación rápida de prototipos para modelar y propagar rápidamente órbitas de satélites, un satélite a la vez. En este bloque se debe especificar el estado orbital inicial y modo de control de actitud (apuntamiento hacia el sol, nadir Tierra o personalizado)

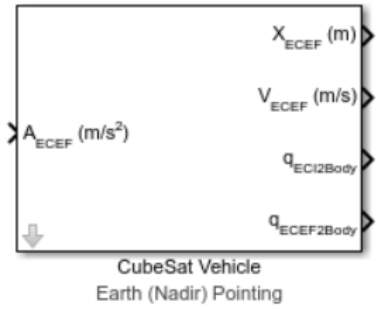


Figura 14. Plantilla de Simulink para el diseño de un *CubeSat*.

- Simulación de Misiones: El proyecto es un ejemplo listo para simular con visualización utilizando Simulink 3D *Animation*, como se muestra en la Figura 15. Este ejemplo utiliza un subsistema de modelo de vehículo por defecto, pero también se puede utilizar el modelo de *CubeSat* diseñado en el paso anterior. Con el módulo, es posible simular misiones completas de *CubeSats*, lo que incluye la evaluación de la trayectoria, la dinámica orbital con modelos de perturbaciones incluidas y las operaciones a bordo. Esto es esencial para prever el comportamiento del *CubeSat* en el espacio y optimizar su rendimiento.

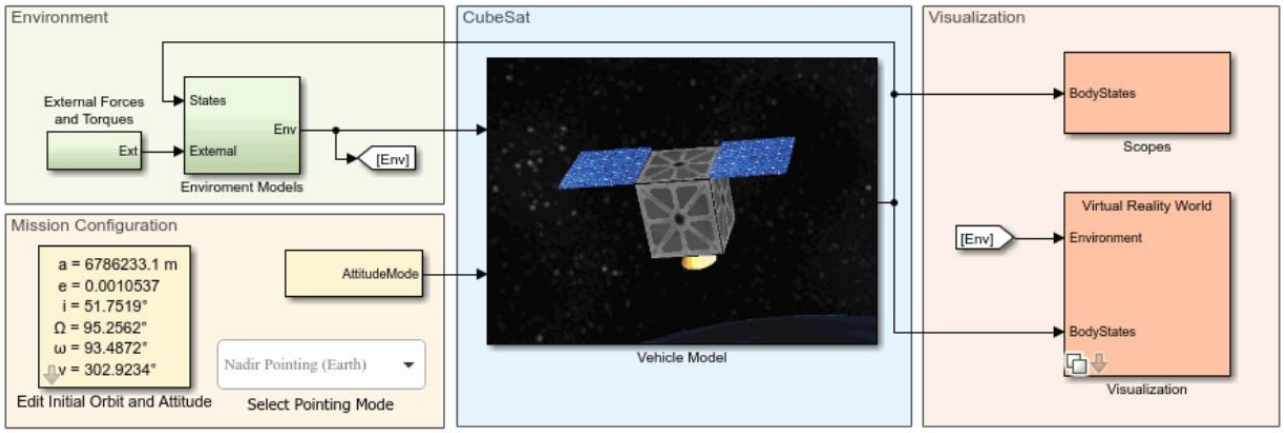


Figura 15. Plantilla de Simulink para la modelación y simulación del *CubeSat* en base a la visualización en Simulink 3D.

- Proyecto de ingeniería de sistemas basados en modelos (MBSE) en *CubeSat*: Es un ejemplo listo para simulación que muestra cómo modelar la arquitectura de una misión espacial con *System Composer* y *Aerospace Blockset*. El proyecto hace referencia al Proyecto de simulación *CubeSat* para reutilizar modelos de subsistema, luego agrega una capa de arquitectura *System Composer*, vincula los requisitos del sistema con los componentes de la arquitectura y verifica los requisitos de la misión de nivel superior con *Simulink Test*™. El proyecto visualiza los resultados utilizando Simulink 3D Animation, escenarios satelitales de *Aerospace Toolbox* y *Mapping Toolbox*™. Todo esto se muestra según los bloques mostrados en la Figura 16.

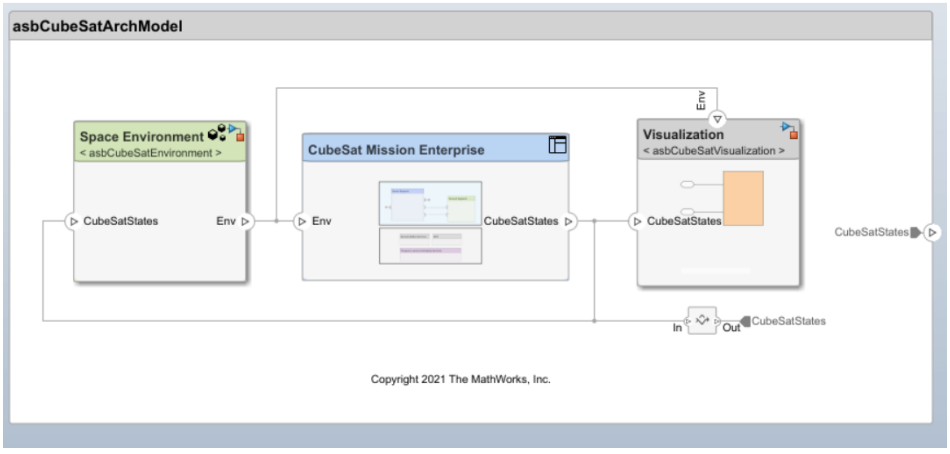


Figura 16. Visualización del MBSE de *CubeSat*.

Mediante la última librería mencionada se puede obtener la observación terrestre de un *CubeSat*, además de visualizar los *budgets* de costo como lo es el precio, la masa, la potencia tanto *peak* como promedio y el tamaño. También presenta en una interfaz gráfica los errores de determinación de actitud y de apuntamiento, además de los torques ejercidos por los actuadores (sean modelados o por defecto dentro del programa), como se muestra en la Figura 17.

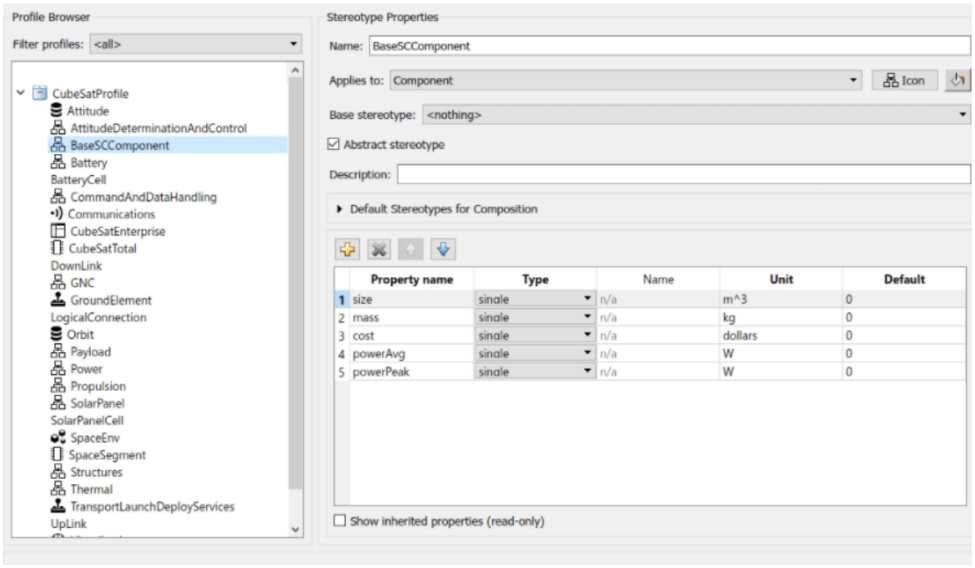


Figura 17. Perfil de *CubeSat* según los SE *envelopes* configurables en base a requisitos impuestos.

Mediante la realización de la ingeniería sistémica basada en modelos de MATLAB con las herramientas correspondientes se puede entregar los siguientes resultados del apuntamiento del satélite gráficamente en la Figura 18.

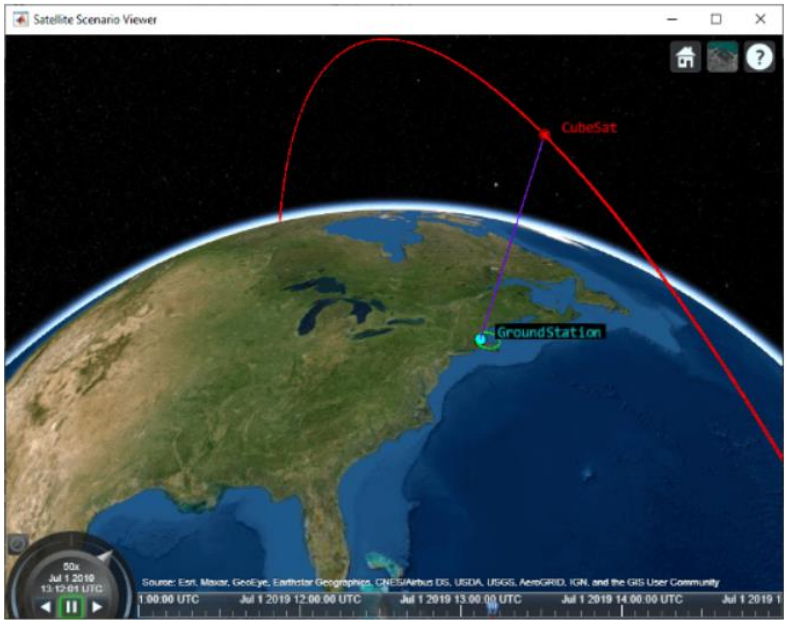


Figura 18. Simulación de apuntamiento de la Tierra hacia la estación terrestre.

3.4 Valispace [33]

Valispace es una plataforma integral de ingeniería y gestión de proyectos diseñada para simplificar y optimizar el proceso de desarrollo de productos y sistemas, especialmente en industrias como la aeroespacial. La plataforma ofrece una variedad de herramientas y características poderosas que permiten a los equipos de ingeniería colaborar eficientemente, gestionar requisitos y parámetros críticos, y tomar decisiones informadas en tiempo real.

Las características claves son la gestión de datos en tiempo real, permitiendo el acceso en tiempo real y la colaboración de los miembros del equipo de ingeniería de forma actualizada. También permite a los equipos diseñar, analizar y optimizar sistemas de ingeniería complejos en base a requisitos que se imponen en el mismo programa al inicio del proyecto. Además, Valispace facilita el cálculo de parámetros críticos como costos, masa, potencia y tamaño, lo que es esencial en proyectos de ingeniería. Los resultados se pueden calcular y actualizar automáticamente a medida que se realizan cambios en el diseño.

En la Figura 19 se muestra la creación de requisitos utilizando como ejemplo la creación de un ventilador en base a la masa máxima de sus componentes. También se puede visualizar el orden jerárquico de los requisitos impuestos, mostrando que la masa de la hélice es un requisito hijo de la masa total.

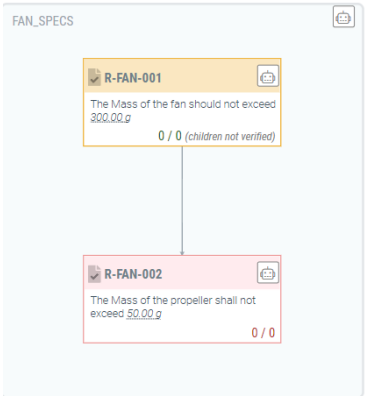


Figura 19. Creación de requisitos respecto a la masa máxima [20].

Por otro lado, en la Figura 20 se visualiza la creación de *valis*, que son parámetros de componentes que contienen sus valores de ingeniería y tiene propiedades como fórmulas, valores, historia y mucho más. Para este caso existen valores agregados a consumo de potencia y masa utilizada, el cual se suma y se verifica posteriormente si cumple con los requisitos impuestos anteriormente.

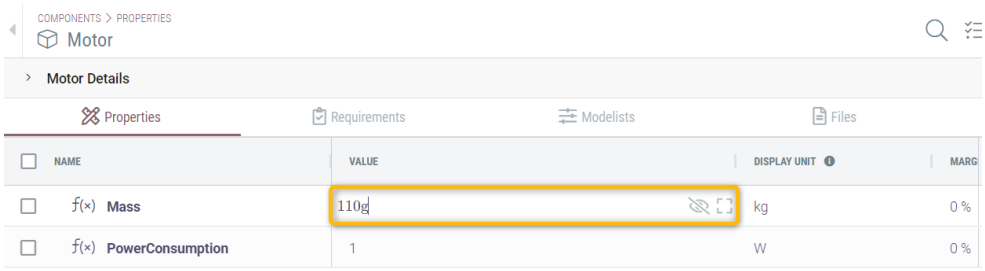


Figura 20. Imposición de valores respecto a la masa y a la potencia consumida [20].

3.5 Resumen de los simuladores disponibles

Para tener en claro si es posible el uso de algunos de estos softwares presentes en el estado del arte (SOA) en la suite de simulación, se toma en cuenta la Tabla 5, en la cual, en base a criterio propios, se analiza cada opción y su viabilidad.

Tabla 5. Comparación entre los simuladores SOA encontrados.

Software/simulador	Precio [USD]	Dificultad	MoP de apuntamiento	SE envelopes
CubeSat Toolbox	495 + Matlab	Media	No presenta	Potencia
SCT académico	3995 + Matlab	Alta	Exactitud de apuntamiento	Potencia y masa
SCT profesional	11995 + Matlab	Alta	Exactitud de apuntamiento/agilidad	Potencia, masa y tamaño
Aerospace Blockset	Matlab	Muy alta	Exactitud de apuntamiento	Todos, incluyendo imposición de requisitos
Systems ToolKit	4527 por año	Media	Exactitud de apuntamiento/agilidad	Potencia
Valispace	1295 por año	Media	No presenta	Todos

Para el caso del precio, son accesibles las opciones de *CubeSat* Toolbox y *Aerospace Blockset*, ya que una suscripción de MATLAB cuesta alrededor de 275 USD por año para versión académica, la cual se presenta disponible en el marco de su uso dentro de la Universidad de Concepción como estudiante/académico. Se descarta la primera opción al no presentar un análisis de los MoP de apuntamiento, ya que solo muestra la potencia gastada y la dinámica orbital y de actitud del satélite mostrada en un modelo 3D hecho en CAD. Otra razón para descartar su uso es que se presenta exclusivamente en MATLAB, siendo que la suite de simulación como se mencionó en las condiciones de diseño se pretende armar en Python, por lo que habría que reformular el código dentro de este lenguaje de programación.

Por otro lado, el *Aerospace Blockset* se presenta como una buena opción, al ser un pack de herramientas disponibles en MATLAB. Si bien esta logra analizar al menos uno de los MoP de apuntamiento que se discutirán en el capítulo 4, su uso se basa en MBSE, el cual requiere de un manejo del *Systems Engineering* de medio a alto, escapándose en parte del conocimiento requerido para el uso del simulador. Además, no presenta la opción de hacer lazo cerrado por fuera del software, por lo que la combinación de herramientas es bastante rígida respecto a lo que ya presenta.

Es por estas razones, que se mantienen los simuladores SCT y Aerospace Blockset solo como una base para implementar de manera propia una suite de simulación capaz de entregar resultados de rendimiento y costo, al armar la base de la dinámica orbital y de actitud en Python desde el inicio.

CAPITULO 4: Rendimiento y costo de apuntamiento

4.1 Cuantificación de los MoP de apuntamiento

Para lograr una capacidad de apuntamiento óptima, se debe considerar los MoP de apuntamiento. El apuntamiento se define como la capacidad que tiene el *CubeSat* para orientar la carga útil hacia un objetivo en específico, el cual, para este proyecto, será la tierra al imponer misiones del tipo observación terrestre. Teniendo esto en cuenta, existen índices de rendimiento con los cuales se verifica si este apuntamiento del satélite fue un éxito o un fracaso. Estos fueron revisados y descritos cualitativamente en [9], y debido a una revisión bibliográfica más extensa en libros sobre diseño de ADCS, se corrigieron y se cuantificaron, quedando como se muestra a continuación:

- Exactitud de apuntamiento [8, 10]: Es el índice con mayor relevancia dentro de las misiones de observación terrestre y de apuntamiento en general. Se refiere al error absoluto de apuntamiento del satélite, por lo que es la capacidad del *CubeSat* de mantener y controlar su orientación hacia una sección especifica de la Tierra. Esta se mide en termino de grados sexagesimales [°] o radianes [rad] y se notará dicho valor como la resta entre la orientación deseada del *CubeSat* y la posición obtenida mediante el ADCS, sabiendo que existen posibles errores de determinación de actitud o de actuadores que no puedan ejercer el torque requerido.
- *Jitter* [8, 34, 35]: El *Jitter* en la línea de visión (LoS, por sus siglas en inglés) de la nave espacial se define como las vibraciones mecánicas sinusoidales de pequeña amplitud que ocurren debido a las interacciones dinámicas causadas por dispositivos mecánicos vibratorios montados en la nave espacial o dentro del instrumento (s) de carga útil y que aparecen a frecuencias en o por encima del ancho de banda del *Attitude Control System* (ACS) del satélite, desde unos pocos Hz hasta cientos de Hz, y que perturban indeseablemente el apuntamiento de la línea de visión de la carga útil. Para visualizar este problema, se debe revisar si existen en las sinusoidales alrededor del eje de equilibrio a controlar en los ángulos de Euler, leves perturbaciones asemejadas al ruido, que representan la inestabilidad del satélite al tomar diferentes orientaciones pequeñas en bajos periodos de tiempo, que hacen que la cámara sea vea “empañada”. Para representar dicho problema, se considera la densidad espectral de potencia como una medida de cuantificación del *jitter*, el cual se obtendrá una vez fijado un filtro pasa alto de hasta una frecuencia adecuada (generalmente 10 [Hz]) aplicada a la respuesta obtenida. Se observará un alto nivel de *jitter* si existe un valor de la densidad espectral de potencia mayor, al consumir más energía en dicho rango de frecuencias.
- Agilidad [8, 36]: Se refiere a lograr una maniobra de actitud mínima, el cual es una combinación de apuntar al objetivo (sección de la Tierra) en el menor tiempo posible a través de una maniobra de giro. En otras palabras, se busca que el *CubeSat* intercepte al objetivo lo más rápido posible y mantenga la orientación en una exactitud de apuntamiento aceptable dentro de los requerimientos. Para cuantificar este índice, se hará uso del parámetro del tiempo de asentamiento (*Settling Time*), el cual especifica el tiempo permitido para recuperarse de maniobras o perturbaciones. Si existe un tiempo de asentamiento menor en alguna comparación de dos ADCS, sería más ágil aquel que

presente un menor tiempo de asentamiento, comparándolo a través de una gráfica de velocidad angular respecto al tiempo, asumiendo una banda de asentamiento adecuada.

- *Drift* [8]: Se define como un límite en el movimiento lento de baja frecuencia de un vehículo. Por lo general, se expresa como [°]/s. Este límite de velocidad angular se deja impuesto para evitar que las perturbaciones cambien la actitud del satélite. Se utiliza cuando el vehículo puede desviarse del objetivo con reinicios poco frecuentes.

Cabe destacar que se eliminó la velocidad de respuesta de los índices de rendimiento de apuntamiento descritos en [9] debido a que es un parámetro que permite cuantificar la agilidad, al presentar similitudes en su definición respecto al tiempo de asentamiento. También notar que en trabajo previo se describió el *drift* como estabilidad de la carga útil, con una definición similar. Se optó por este nombre debido a que es el que se suele utilizar para analizar las implicancias de un exceso de perturbaciones sobre el satélite a través de una velocidad angular menor a la requerida previo a la misión.

4.2 Influencia del ADCS sobre los MoP de apuntamiento

En primera instancia se hará notar como afectan tanto el software como el hardware del ADCS a los MoP de apuntamiento. Esto se describe en la Tabla 6 mostrada a continuación:

Tabla 6. Influencia del ADCS sobre los MoP de apuntamiento [8, 9].

Parte del ADCS	MoP de apuntamiento que afecta	Descripción
Sensores	Exactitud de apuntamiento *Jitter* *Drift*	Los sensores afectan directamente a la exactitud de apuntamiento y al *jitter* dependiendo de la calidad de estos al momento de determinar la actitud del satélite (mayor calidad es menor error en la determinación de actitud y menor ruido). Por otro lado, para el caso del *drift*, se requiere un conocimiento óptimo de la velocidad angular otorgada por el giroscopio de manera tal de analizar si existe o no este fenómeno.
Algoritmos de determinación de actitud	Exactitud de apuntamiento *Jitter*	Dependiendo de la calidad del algoritmo de estimación utilizado, se puede obtener un mayor o menor error absoluto de apuntamiento. Por otro lado, la utilización de un mejor algoritmo que disminuya el ruido de los sensores hace que mejore el *jitter* presente en los gráficos.

Controladores	Exactitud de apuntamiento *Jitter* *Drift* Agilidad	Dependiendo de la calidad del controlador a implementar en el satélite, se puede disminuir el tiempo de asentamiento, provocando maniobras más rápidas y estables. Por otro lado, si el controlador otorga ganancias excesivas, haciendo que el actuador gire con alta velocidad, puede provocar *jitter* en la línea de visión al generar vibraciones que se denotarían en los gráficos de orientación respecto al tiempo. También son los responsables de generar una velocidad angular sobre la mínima para evitar el *drift*. Además, se logra obtener una determinada exactitud dependiendo del controlador a utilizar y de las constantes ejercidas para su implementación, oscilando alrededor de un *setpoint* diferente al requerido o estabilizándose inmediatamente.
Actuadores	Exactitud de apuntamiento *Jitter* *Drift* Agilidad	Los actuadores están directamente relacionados con la exactitud de apuntamiento, ya que, dependiendo de la sensibilidad rotacional del satélite ejercida por el torque del actuador, puede tener una mayor exactitud de apuntamiento al corregir la orientación con torques más pequeños. Por otro lado, dependiendo del actuador utilizado puede mejorar la agilidad, ya que, si se usa un componente capaz de generar mayor torque, esto reduce el tiempo de asentamiento, siempre y cuando el controlador sea capaz de estabilizarlo hacia la orientación deseada. Bajo este mismo criterio, el actuador puede ayudar a mantener una cierta velocidad angular para evitar el *drift*. Por otro lado, un control inadecuado del actuador hará que existan vibraciones mecánicas internas excesivas que generaran un aumento del *jitter* en el sistema, representándose en oscilaciones y ruido en las orientaciones.

4.3 Costo de los componentes de ADCS según los SE *envelopes*

Con el objetivo de visualizar en el simulador el rendimiento de apuntamiento y su costo asociado, se decide agrupar los componentes físico en niveles según el COTS disponible en *CubeSat* [21]. Esto se hace debido a que existen una variedad considerable de componentes capaces de ser usado en este tipo de nanosatélites, al ser de poco costo tanto energético como monetario. Caracterizar todos estos se escapa de los objetivos del trabajo, ya que su consideración genera un análisis extenso e innecesario respecto a la obtención de los costos en base a los SE *envelopes*. Por ello, se presenta en la Tabla 7 los componentes COTS seleccionados en conjunto con su proveedor para analizar tanto el rendimiento

(MoP de apuntamiento) como el costo utilizado por cada uno de ellos. Mas detalles de cada uno de los sensores y actuadores elegidos en el Anexo C.

Tabla 7. Componentes elegidos a representar para cada nivel [21].

Componente	Nivel bajo	Nivel medio	Nivel alto
GPS receiver	Explorer (General Dynamics)	SGR-Ligo (Surrey Satellite Technology)	Celeste_gnss_rx (Spacemanic)
Giroscopio	CRH03 – 200 (Silicon Sensing Systems)	CRH03 – 010 (Silicon Sensing Systems)	NSGY-001 (NewSpace System)
Magnetómetro	MM200 (AAC Clyde Space)	-	MAG-3 (AAC Clyde Space)
Sun Sensor	CSS-01, CSS-02 (Space Micro)	MSS-01(Space Micro)	FSS (Bradford Space)
Rueda de reacción	CubeWheel Small (CubesSpace)	RW4-1.0 (RocketLab)	RW4 (Blue Canyon Technologies)
Magnetorquer	CubeTorquer Coil (CubeSpace)	NCTR-M012 (NewSpace systems)	MT15-1 (ZARM Technik)

CAPITULO 5: Diseño de la suite de simulación

5.1 Marco general del simulador

Para la construcción de la suite de simulación, se debe tener en cuenta tanto la dinámica orbital como la dinámica de actitud, para así tener conocimiento del posicionamiento y la velocidad del *CubeSat* a través del espacio, y de la actitud de este en su desplazamiento alrededor de la Tierra. Para ello se debe considerar los siguientes elementos como base:

- Propagación orbital: En primera instancia, el simulador busca conocer el posicionamiento del satélite alrededor de la Tierra, por lo que es necesario el uso de un propagador orbital adecuado capaz de entregar información sobre el vector posición y velocidad del satélite respecto a la Tierra. Además, se debe considerar las perturbaciones correspondientes para la altura del satélite analizado, que en este caso se acotará a órbitas bajas, con el objetivo de hacer el movimiento traslacional del satélite realista según las condiciones espaciales presentes.
- Modelos orbitales: Los modelos orbitales son necesarios para obtener la orientación del *CubeSat*. Con ello se obtendrán las representaciones necesarias de los sensores en el marco de referencia inercial.
- Sensores: Para determinar la actitud del satélite, se seleccionan sensores capaces de entregar información relevante sobre la inercia del satélite, el comportamiento físico del ambiente espacial o estrellas circundantes. Es relevante tanto para la estimación inicial como para el conocimiento de la actitud a través del tiempo y se simularan según los modelos orbitales analizados, teniendo en cuenta una rotación del marco de referencia inercial al marco de referencia del cuerpo y el ruido en la lectura de los componentes utilizados.
- Algoritmos de estimación de actitud: Estos son necesarios para trabajar los vectores entregados por los sensores y los modelos orbitales. Con estos se obtienen los cuaterniones que representan las rotaciones del satélite a través del tiempo. Se busca utilizar el TRIAD para la estimación inicial y el EKF para la estimación de los estados posteriores, además de la eliminación del ruido en la obtención de la orientación del satélite.
- Controladores y actuadores: Ya obtenidas la actitud del satélite a través del tiempo, se desea que el *CubeSat* apunte a una dirección en específico. Para ello se implementa el uso de un controlador capaz de realizar el control del satélite mediante la entrega del modelo dinámico y del actuador. Esto va de la mano con la selección del actuador a utilizar, para tener en cuenta las limitaciones de la acción de control al apuntar el satélite hacia la dirección deseada.

Teniendo en cuenta estos factores, se muestra en la Figura 21 un diagrama que resume la base de la suite de simulación para la posterior obtención de resultados de los MoP de apuntamiento y de los SE *envelopes* correspondientes. Se observa en primera instancia la obtención del vector estado $\vec{\boldsymbol{r}}$ del *CubeSat* mediante el propagador orbital, para posteriormente utilizar los modelos orbitales y conocer V_{ref} que es el vector en el sistema de referencia inercial. Además, mediante la lectura de los sensores se conocerá V_{obs}, que representan los vectores obtenidos en el marco de referencia del cuerpo. Teniendo en cuenta el cambio de sistema de referencia de V_{ref} a LVLH (que se explicará en las siguientes secciones), se utilizan ambos vectores para la estimación del cuaternión q, para finalmente

ejecutar la acción de control con el controlador seleccionado, teniendo la restricción del torque $\boldsymbol{\tau}$ ejercido por el actuador. En las siguientes secciones se explicará con mayor detalle el diseño/selección de cada parte de la suite de simulación.

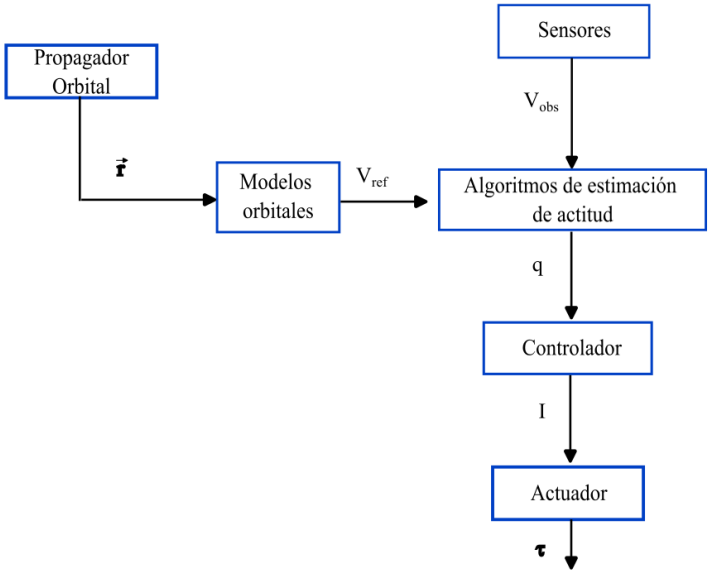


Figura 21. Esquema general de la base del simulador.

5.2 Propagador orbital

Para la simulación de la dinámica orbital, se debe considerar la implementación de un propagador capaz de entregar la posición y velocidad del satélite respecto a la Tierra incluyendo las perturbaciones relevantes a baja altura. Para ello se proponen las siguientes opciones:

Simplified General Perturbation 4 (SGP4) [37]:Es un modelo matemático que se utiliza ampliamente para predecir la posición y velocidad de los satélites. Utiliza las ecuaciones de movimiento de dos cuerpos, que describen cómo un satélite se mueve bajo la influencia de la gravedad de la Tierra, y luego aplica una serie de perturbaciones para tener en cuenta factores como la forma no esférica de la Tierra, el arrastre atmosférico, la radiación solar y los efectos gravitacionales de la interacción con otros cuerpos como el Sol o la Luna. El SGP4 utiliza los *Two Line Elements* (TLE) como entrada, que es un formato estándar utilizado para describir la órbita de un satélite en dos líneas, en conjunto con la fecha de inicio y el tiempo de propagación, para después aplicar una serie de ecuaciones y algoritmos que calculan los elementos orbitales futuros del satélite en función del tiempo.

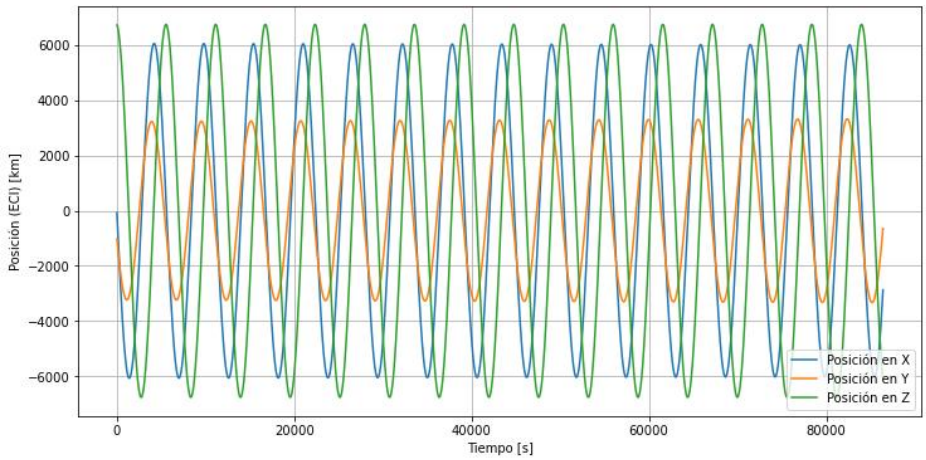
FreeFlyer [38]:Es un programa de simulación espacial diseñado para visualizar y modelar varios escenarios, incluyendo, pero no limitándose a la propagación y maniobras de naves espaciales, análisis de cobertura y contacto, análisis interplanetario y la generación de diversos recursos visuales. Es capaz de propagar el movimiento del satélite a través del tiempo e incluir perturbaciones como J2, arrastre atmosférico, efecto multi-cuerpo y radiación solar.

Systems Tool Kit (STK) [32]: Es una plataforma ampliamente utilizada en el ámbito tanto aeronáutico como espacial. El STK es un simulador de Ansys utilizado para el diseño, planificación y simulación

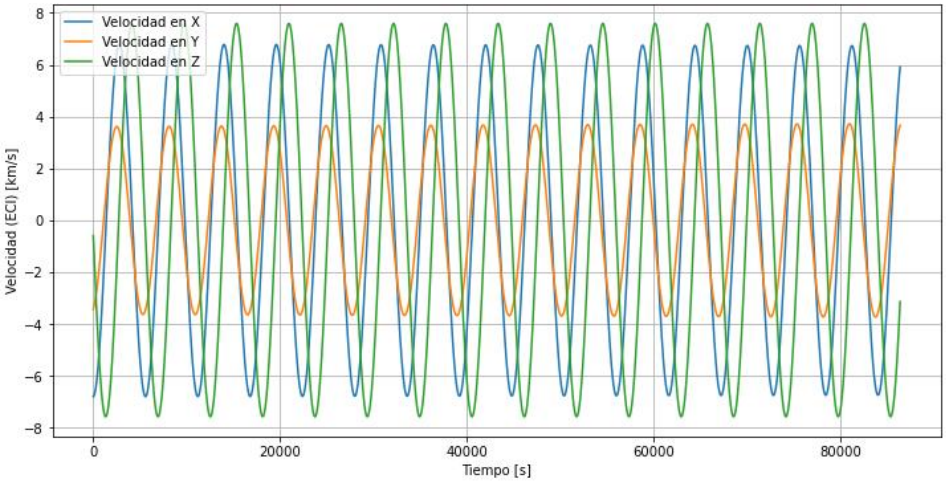
de misiones. Una de sus herramientas es la propagación del satélite a través del tiempo con una interfaz gráfica, incluyendo todas las perturbaciones presentes en el espacio.

Se descartan las opciones de los softwares *FreeFlyer* y STK debido a su alto costo monetario para la implementación del simulador solo para su uso en la dinámica orbital. Por lo tanto, se elige el SGP4 no solo por ser una opción gratuita, sino porque es fácil de implementar a cualquier satélite que se conozca su TLE. Esta entrega el movimiento traslacional del satélite a través del tiempo considerando las perturbaciones más importantes a bajas alturas.

Para analizar el funcionamiento del propagador, se utilizó como ejemplo un TLE del SUCHAI-3 obtenido de CelesTrak y se propagó durante un día tomando como fecha de inicio 1 de noviembre del 2023 a las 12 del mediodía. Se observa en la Figura 22 (a) y (b) la posición y la velocidad respecto al marco de referencia ECI obtenido en Python a través de la librería SGP4.



(a)



(b)

Figura 22. Propagación del SUCHAI-3 durante un día con fecha de inicio 01/11/23 para (a) posición (b) velocidad.

5.3 Modelos orbitales y sistema de referencia.

Para la obtención de la actitud del satélite se requieren conocer dos vectores respecto al marco de referencia Local Vertical Local Horizontal (LVLH) o también llamado "orbital". Para ello se conocerán en primera instancia los modelos del sol y del campo geomagnético terrestre para la obtención de los vectores en el marco de referencia inercial (ECI), para luego rotarlos al sistema LVLH. Posteriormente se obtendrán los vectores de observación, que se generan en base a los modelos mencionados aplicando otra rotación desde orbital al cuerpo, que representan los vectores medidos por los sensores.

5.3.1 Vector sol [15]

El modelo se basa principalmente en el movimiento del sol respecto al sistema de referencia ECI. Lo primero a tener en cuenta es la anomalía media del sol, la cual se calcula según muestra la Ecuación (5) y consiste en el ángulo medido desde el perigeo, que describe la posición del Sol en su trayectoria orbital y depende puramente del tiempo.

$$M_{sun} = M_{sunEpoch} + n_{sun} JD_{2000} = 357.528° + 0.9856003° \mathrm{JD}_{2000} \tag{5}$$

En la ecuación anterior, $M_{sunEpoch}$ es el valor conocido de la anomalía media del Sol para el 1 de enero de 2000 al mediodía en UTC y n_{sun} es el movimiento medio del Sol, el cual también es un valor conocido. La anomalía media del Sol puede entonces calcularse para cualquier JD_{2000}, que se refiere a la fecha juliana respecto al año 2000. A partir de la anomalía media, se puede calcular la longitud de la eclíptica λ_{sun} según la siguiente ecuación, que representa la posición del Sol en el plano orbital bidimensional en el marco ECI.

$$\lambda_{sun} = 280.461° + 0.9856474 JD_{2000} + 1.915° \sin(M_{sun}) + 0.020° \sin(2M_{sun})$$

Para determinar completamente la posición del Sol en el marco ECI, se requiere conocimiento de la inclinación de la órbita desde el plano orbital bidimensional. Este parámetro se llama oblicuidad del plano de la eclíptica y se define a continuación:

$$\varepsilon = 23.4393° + 0.0000004° \, \mathrm{JD}_{2000}$$

La posición del Sol en el marco ECI ahora se puede calcular como un vector unitario con las componentes representadas en la Ecuación (6), (7) y (8)

$$X_{sun} = \cos(\lambda_{sun}) \tag{6}$$

$$Y_{sun} = \cos(\varepsilon) \sin(\lambda_{sun}) \tag{7}$$

$$Z_{sun} = \sin(\varepsilon) \sin(\lambda_{sun}) \tag{8}$$

Este modelo del vector sol se observa en la Figura 23, en la cual se simula propagando por un año desde el 05 de julio del 2023, el cual en días julianos corresponde al día 8586. Se observa en dicha gráfica un comportamiento oscilatorio, notándose un ciclo entero del movimiento del sol recién al término de la simulación.

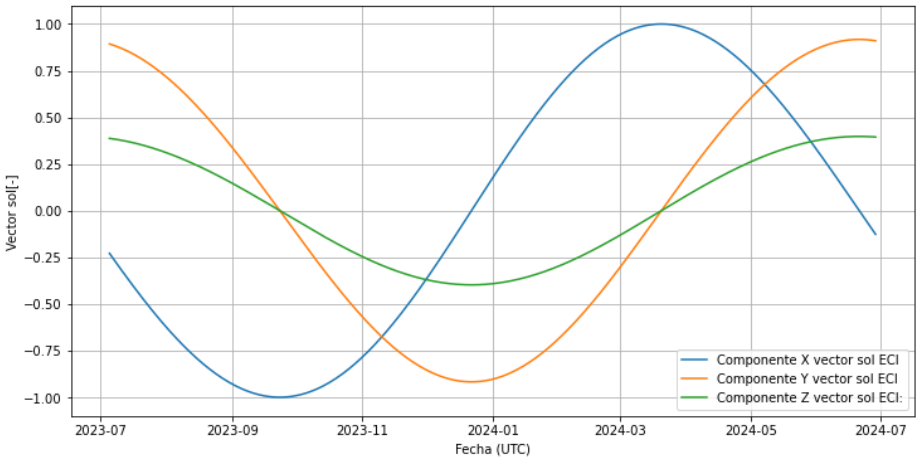


Figura 23. Vector sol en sus tres componentes simulados durante un año.

5.3.2 Fuerzas geomagnéticas de la Tierra [39]

Para el modelamiento de las fuerzas geomagnéticas de la tierra en distintas posiciones de la órbita de un satélite se requiere el uso del *International Geomagnetic Reference Field* (IGRF). Este campo de referencia consiste en un conjunto de coeficientes armónicos esféricos que pueden introducirse en un modelo matemático, para así describir la porción a gran escala y variable en el tiempo del campo magnético interno de la Tierra entre las épocas 1900 d.C. y el presente. El IGRF utilizado para la realización del simulador es el de decima tercera generación y se ha derivado de observaciones registradas por satélites, observatorios terrestres y estudios magnéticos

El IGRF describe el principal campo geomagnético $B(r,\theta,\phi,t)$ que es producido por fuentes internas principalmente dentro del núcleo de la Tierra. El IGRF es válido dentro y alrededor de la superficie de la Tierra, donde el campo geomagnético principal puede describirse como el gradiente de un potencial escalar, $B = -\nabla V$, y la función potencial $V(r,\theta,\phi,t)$ se representa en la Ecuación (9) como una expansión en serie finita en términos de coeficientes armónicos esféricos, g_n^m, h_n^m, también conocidos como coeficientes de Gauss:

$$V(r,\theta,\phi,t) = a\sum_{n=1}^{N}\sum_{m=0}^{n}\left(\frac{a}{r}\right)^{n+1}[g_n^m(t)\cos(m\phi) + h_n^m(t)\sin(m\phi)]P_n^m(\cos(\theta)) \qquad (9)$$

Aquí, r,θ,ϕ se refieren a coordenadas en un sistema de coordenadas esféricas geocéntricas o geodésicas, siendo r la distancia radial desde el centro de la Tierra o la altura desde su superficie dependiendo de si son geocéntricas o geodésicas respectivamente, y θ,ϕ son la co-latitud y longitud geocéntricas respectivamente. El $P_n^m(\cos(\theta))$ son funciones de Legendre asociadas seminormalizadas de Schmidt de grado n y orden m. El parámetro N especifica el grado máximo de expansión armónica esférica y se eligió que fuera 10 hasta la época 1995 inclusive, después de lo cual aumenta a 13. Por otro lado, los coeficientes de Gauss cambian en el tiempo y se proporcionan en unidades de nanoTesla (nT) en IGRF-13 en intervalos de época de 5 años. La dependencia temporal de estos parámetros se modela como lineal por partes y viene dada por:

$$g_n^m(t) = g_n^m(T_t) + (t - T_t)\dot{g}_n^m(T_t)$$

$$h_n^m(t) = h_n^m(T_t) + (t - T_t)\dot{h}_n^m(T_t)$$

Dónde $g_n^m(T_t)$, $h_n^m(T_t)$ son los coeficientes de Gauss en la época T_t, que precede inmediatamente al tiempo t. Las épocas del modelo en IGRF-13 se proporcionan en múltiplos exactos de 5 años comenzando en 1900 y terminando en 2020, de modo que $T_t < t < T_t + 5$. Para $T_t < 2020$, los parámetros $\dot{g}_n^m(T_t)$, $\dot{h}_n^m(T_t)$ representan la aproximación lineal al cambio en los coeficientes de Gauss durante el intervalo de 5 años que abarca $[T_t, T_t + 5]$. Pueden calcularse en unidades de nanoTesla por año (nT/año) como:

$$\dot{g}_n^m(T_t) = \frac{1}{5}\left(g_n^m(T_t + 5) - g_n^m(T_t)\right)$$

$$\dot{h}_n^m(T_t) = \frac{1}{5}\left(h_n^m(T_t + 5) - h_n^m(T_t)\right)$$

Este procedimiento en Python viene entregado por la National Centers for Environmental Information (NCEI) [40] bajo el nombre de paquete *py.igrf* y se implementa en el simulador para el conocimiento de las fuerzas magnéticas ejercidas sobre la Tierra. Se requiere para el modelo insertar como parámetros de entrada las coordenadas geodésicas (r, θ, ϕ) en conjunto con el año de análisis, para así obtener siete salidas (declinación [°], inclinación [°], intensidad horizontal [nT], componente norte [nT], componente este [nT], componente vertical [nT] e intensidad total [nT]), siendo la componente 4,5 y 6 las fuerzas magnéticas en el marco de referencia ECI.

Para conocer las coordenadas geodésicas y simular un receptor GPS sin ruido dentro del satélite, se hizo un cambio de sistema de referencia desde ECI a geodésicas. Utilizando la librería Skyfield de Python y conociendo la posición en la cual está el satélite respecto a la Tierra gracias al propagador SGP4, se conocen las coordenadas GPS en cada instante del movimiento traslacional del satélite. Para analizar el funcionamiento del modelo IGRF entregado en Python, se utilizó los vectores posición y velocidad de la sección 5.3.1 correspondiente al SUCHAI-3 para hacer el cambio de sistema ECI a geodésicas, obteniendo en la Figura 24 las fuerzas magnéticas del satélite a través del tiempo.

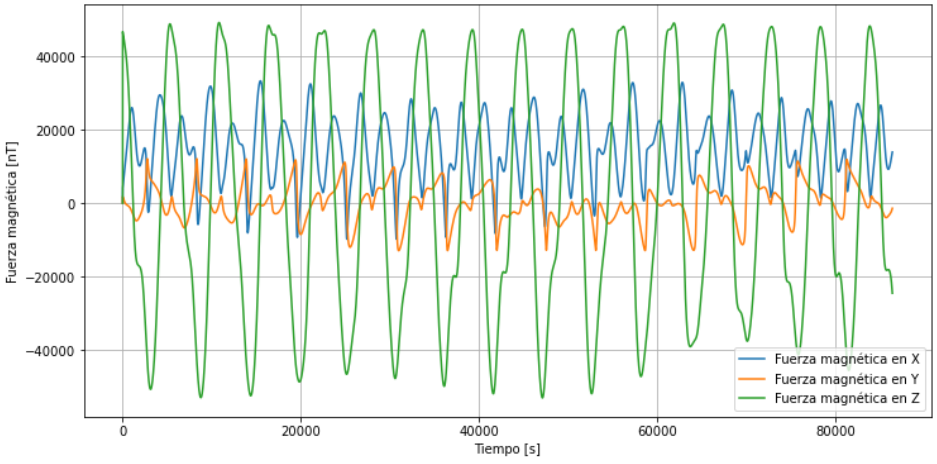


Figura 24. Componentes de las fuerzas magnéticas respecto a ECI que afectan al SUCHAI-3 con fecha inicial 01/11/2023.

5.3.3 Cambios en los sistemas de referencia y vectores de observación

Ya obtenidos los vectores sol y las fuerzas magnéticas en ECI, se aplica una matriz de rotación desde ECI a LVLH utilizando los vectores posición $\vec{P}$ y velocidad $\vec{V}$ calculados en la sección 5.2 proveniente del SGP4. Dicha matriz de rotación se representa de la siguiente manera [35]:

$$\overrightarrow{Z_O} = \frac{\vec{P}}{\|\vec{P}\|} \quad ; \quad \overrightarrow{X_O} = \frac{\vec{V} \times \overrightarrow{Z_O}}{\|\vec{V} \times \overrightarrow{Z_O}\|} \quad ; \quad \overrightarrow{Y_O} = \overrightarrow{Z_O} \times \overrightarrow{X_O}$$

$$R_{ECI}^{O} = \left[\overrightarrow{X_O}^T \quad \overrightarrow{Y_O}^T \quad \overrightarrow{Z_O}^T\right]$$

Utilizando esta matriz, en conjunto con el DCM obtenido por el algoritmo TRIAD, la cual se representa como R_{ECI}^{body}, se obtiene la matriz de rotación que relaciona el marco de referencia del cuerpo con el orbital, mediante la siguiente relación.

$$R_{O}^{body} = (R_{ECI}^{O})^T \cdot R_{ECI}^{body}$$

Mediante esta matriz de rotación, la cual puede ser transformada a un cuaternión q_O^{body}, se puede aplicar el control que será analizado en secciones posteriores utilizando una condición inicial en el sistema de referencia adecuado. Para la obtención de los vectores sol y las fuerzas magnéticas en el marco de referencia del cuerpo, se genera el cuaternión entre los marcos de referencia a medida que se estima en el siguiente paso de tiempo. Es relevante reconocer que se utilizan las mismas relaciones de rotación para la obtención de los vectores B_O y $v_{sun,O}$ usando los cuaterniones q_{ECI}^{O}.

$$\left(q_O^{body}\right)^{-1} = [-q_0, -q_1, -q_2, q_3]$$

$$B_{body} = (q_O^{body})^{-1} \cdot B_O \cdot q_O^{body}$$

$$v_{sun,body} = (q_O^{body})^{-1} \cdot v_{sun,O} \cdot q_O^{body}$$

5.4 Algoritmos de estimación y control satelital

Ya obtenidos los vectores de referencia y de observación, se puede estimar la orientación (cuaterniones) y la velocidad angular del satélite a través del tiempo. Para ello se estima la actitud actual mediante mediciones y métodos matemáticos, para posteriormente definir una acción de control que logrará apuntar hacia la actitud objetivo (sistema de referencia LVLH) utilizando un sistema de control automático.

5.4.1 TRIAD y EKF

En primera instancia se utiliza el método determinista del TRIAD para obtener la orientación inicial del satélite, en el cual los vectores V_1 y V_2 serían las mediciones de los magnetómetros y los sensores de sol respectivamente, mientras que los vectores W_1 y W_2 son los modelos orbitales del campo geomagnético de la tierra y el modelo del sol en el marco de referencia ECI. Utilizando las ecuaciones de la sección 2.4.2 del enfoque determinista, se tiene la siguiente matriz de rotación:

$$R = M_{obs}\left(M_{ref}\right)^{T}$$

Es relevante mencionar que el magnetómetro se toma siempre como el V_1 debido a que es mas exacto en la determinación de la fuerza magnética respecto al sensor de sol en su obtención del vector sol en los diferentes niveles. Además, se destaca que la solución entrega una matriz de rotación (Direction Cosine Matrix) entre los marcos de referencia ECI y del cuerpo, por lo tanto, se debe convertir a orbital/cuerpo mediante la relación previamente mostrada en 5.3.3, para posteriormente transformarla a cuaternión para su trabajo posterior. Además, para el análisis de resultados, se utilizan los ángulos de Euler para una mayor comprensión conceptual y física de los cambios de orientación ocurridos en el satélite. En el Anexo D se presentan dichas conversiones matemáticas.

Se descarta en su totalidad el uso del TRIAD para la determinación de la orientación del *CubeSat* en cada paso de tiempo, debido a que al ser un algoritmo simple y a la vez primitivo, tiene un alto error en la estimación de actitud respecto a la orientación ideal, haciendo que en la sección posterior de control no se logre llegar al *setpoint* deseado. Mas detalles se muestran en la sección 6.1, al mostrar resultados utilizando datos e información de un *CubeSat*.

Por otro lado, se usa generalmente el método recursivo como el filtro extendido de Kalman (EKF) para estimar la actitud del satélite con una mayor exactitud y menores costos computacionales, utilizando como orientación inicial el cuaternión proveniente del TRIAD. Para la elaboración de este trabajo, se consideró la estimación del EKF mediante la asignación de un error aleatorio equivalente al 10% del cuaternión ideal para representar dicho algoritmo de manera simple y acotada, por lo que queda pendiente su implementación real dentro del simulador por falta de tiempo. Para conocer en mayor detalle el avance hecho en el filtro de Kalman extendido se recomienda revisar el Anexo E donde se muestra el paso a paso y las consideraciones tomadas para su implementación.

5.4.2 Controlador PD y actuador [14]

Para el controlador, se opta por el uso del Proporcional-Derivativo (PD) debido a su simplicidad y que cumple con la función de dirigir el satélite en una orientación especifica. También se opta por el uso solo del magnetorquer como actuador debido a la documentación encontrada sobre su utilización y al acotamiento del tiempo de trabajo, cuyo torque de control se representa de la siguiente manera en sus tres componentes [14]:

$$\tau_{x,ctrl} = \left(\frac{b_x m_z - b_z m_x}{\|b\|}\right) b_z - \left(\frac{b_y m_x - b_x m_y}{\|b\|}\right) b_y$$

$$\tau_{y,ctrl} = -\left(\frac{b_z m_y - b_y m_z}{\|b\|}\right) b_z + \left(\frac{b_y m_x - b_x m_y}{\|b\|}\right) b_x$$

$$\tau_{z,ctrl} = \left(\frac{b_z m_y - b_y m_z}{\|b\|}\right) b_y - \left(\frac{b_x m_z - b_z m_x}{\|b\|}\right) b_x$$

Donde *b* son las fuerzas magnéticas en el marco de referencia del cuerpo y m es el momento dipolar del magnetorquer.

Para utilizar el controlador, es fundamental comprender la linealización del sistema, un proceso útil para simplificar modelos no lineales como las ecuaciones (2) y (4) y facilitar el diseño de estrategias de control. Este proceso se basa en la aproximación de Taylor [41], que consiste en linealizar las ecuaciones alrededor de un punto de equilibrio, cuyos valores en este caso representan que no existe diferencia de orientación entre los marcos de referencia del cuerpo y orbital, además de lograr que se minimice tanto la acción de control como la velocidad angular [14].

$$q_{co} = [0,0,0,1]; u = [0,0,0]; \omega_{co} = [0,0,0]$$

Una vez identificado el punto de equilibrio, se aplica la aproximación de Taylor para linealizar las ecuaciones del sistema alrededor de este punto [41]. Esto implica calcular las derivadas parciales de las funciones de estado con respecto a las variables de estado y de control. Las matrices resultantes, $A = \frac{\partial f(x,u)}{\partial x}$ y $B = \frac{\partial f(x,u)}{\partial u}$, representan la dinámica linealizada del sistema, generando así el modelo espacio estado (EE) mostrado a continuación.

$$\dot{x} = Ax - Bu$$

Es importante destacar que la linealización nos proporciona una descripción local del comportamiento del sistema alrededor del punto de equilibrio, y por lo tanto, la validez de este enfoque depende de la proximidad del estado actual del sistema al punto de equilibrio. Para conocer el álgebra y las consideraciones del diseño del control lineal a mayor detalle, se recomienda revisar el Anexo F.

Evaluando los valores deseados de q_{co} y ω_{co} a las matrices A y B mencionadas en el Anexo F, se obtiene:

$$A = F = \begin{bmatrix} 0 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \\ 6\omega_{0,o}^2[I_z - I_y] & 0 & 0 & 0 & 0 & 0 \\ 0 & 6\omega_{0,o}^2[I_z - I_x] & 0 & 0 & 0 & (\omega_{0,o})(I_x - I_z)/I_y + I_y\omega_{0,o} \\ 0 & 0 & 0 & 0 & (\omega_{0,o})(I_x - I_y)/I_z - I_z\omega_{0,o} & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{-b_z^2 - b_y^2}{\|b\|I_x} & \frac{b_x b_y}{\|b\|I_x} & \frac{b_x b_z}{\|b\|I_x} \\ \frac{b_x b_y}{\|b\|I_y} & \frac{-b_z^2 - b_x^2}{\|b\|I_y} & \frac{b_y b_z}{\|b\|I_y} \\ \frac{b_x b_z}{\|b\|I_z} & \frac{b_y b_z}{\|b\|I_z} & \frac{-b_y^2 - b_x^2}{\|b\|I_z} \end{bmatrix}$$

Ya conseguidas las matrices A y B, se puede representar el modelo de espacio de estados de manera lineal según se muestra a continuación:

$$\dot{x} = \hat{A}x$$

Con:

$$\hat{A} = A - BK$$

$$K = \begin{bmatrix} K_{p,x} & 0 & 0 & K_{d,x} & 0 & 0 \\ 0 & K_{p,y} & 0 & 0 & K_{d,y} & 0 \\ 0 & 0 & K_{p,z} & 0 & 0 & K_{d,z} \end{bmatrix}$$

Donde K_p y K_d son las constantes proporcionales y derivativas del controlador y desempeñan un papel crucial en la determinación de la estabilidad del sistema. Estas constantes se ajustan de manera que los valores propios de la matriz $A - BK$ sean negativos, lo que garantiza la estabilidad del sistema en lazo cerrado.

Los valores propios negativos son indicativos de que el sistema es asintóticamente estable, lo que significa que las respuestas transitorias del sistema convergen hacia el estado de equilibrio deseado sin oscilaciones ni divergencias. En otras palabras, los valores propios negativos aseguran que el sistema retorne a su estado de equilibrio después de cualquier perturbación, lo que es fundamental para el funcionamiento adecuado del sistema de control, con un rendimiento estable y robusto [42].

Además, es relevante mencionar que se utiliza el estado estimado por el algoritmo de estimación EKF para la acción de control $u = Kx$, mientras que el estado ideal está presente multiplicando la matriz A, mostrando así que la estimación no generará el control necesario para estabilizar el sistema y que por lo tanto existirá un margen de error en la obtención de resultados de rendimiento en base a los MoP de apuntamiento.

5.5 Suite de simulación completa

Ya efectuado lo necesario para la base del simulador, se hace un breve resumen sobre las decisiones tomadas para el simulador de manera ideal:

- Uso del propagador SGP4 para la simulación de la dinámica orbital: Se eligió este debido a que presenta disponibilidad en el lenguaje de programación Python, además de ser de libre acceso y modela las perturbaciones necesarias en LEO a través del tiempo. Los demás propagadores como *FreeFlyer* y STK mencionados no son de libre acceso (tienen un alto costo monetario) y pueden no tener una conexión directa con las demás herramientas que se implementaran dentro del diseño del ADCS.
- Modelos orbitales y sensores por utilizar: En el caso de los modelos orbitales, se logra implementar los modelos de la fuerza geomagnética y del vector sol con éxito. Por otro lado, sabiendo los modelos orbitales seleccionados, se elige simular el magnetómetro y el sensor de sol, debido a que pueden obtener los vectores de observación a través de un cambio en el sistema de referencia de cuerpo a orbital, además de la implementación del ruido característico del componente físico. Finalmente, el giroscopio aporta con la actualización del cambio angular en el *CubeSat*, por lo que ayuda a obtener las velocidades angulares a través del tiempo.
- Algoritmo de determinación de actitud seleccionado: Al observar una mala estimación del TRIAD, solo se le utilizó como estimación inicial. Además, se le impuso para tener una primera

iteración del simulador un error representativo al EKF, con el objetivo de observar cómo se comporta el control del satélite mediante niveles de error correspondiente tanto al algoritmo como a los sensores.

- Actuador y controlador por utilizar: Como actuador se simulará solo los magnetorquers para el control de la actitud en el simulador debido a que existen mayor cantidad de referencias sobre su simulación en software [15] [43]. Por otro lado, se elige la utilización de un controlador Proporcional-Derivativo (PD) debido a que la dinámica de actitud del satélite se puede representar como un cuaternión (elemento proporcional) y la velocidad angular (elemento derivativo), siendo suficiente para dar la orden de controlar la actitud de satélite hacia una dirección deseada.

Ya tomadas las decisiones, se puede visualizar el esquema deseado del ADCS en la Figura 25. En el diagrama se muestra que el satélite deberá determinar en primera instancia su posición y velocidad en la órbita. Para ello, se le otorga al propagador parámetros de entrada como el TLE del satélite, una fecha de simulación y un tiempo de propagación. Luego, se simulará el GPS sin error, que determina la ubicación actual en coordenadas geodésicas, haciendo un cambio en el sistema de referencia desde ECI (marco de referencia con el cual se obtiene la posición utilizando el SGP4). Esta información permitirá que se obtengan los valores del campo magnético para la ubicación actual del satélite en órbita en conjunto con el vector del sol, utilizando los modelos orbitales correspondientes.

Por otro lado, el *CubeSat* obtendrá los vectores de observación mediante el uso de los sensores de sol y los magnetómetros, obteniendo lecturas en el marco de referencia del cuerpo y simulándolo mediante una rotación de los modelos orbitales utilizando los cuaterniones que se obtengan del control satelital, además de la simulación del giroscopio como el sensor capaz de obtener la tasa de cambio de la actitud del satélite. Posteriormente, se conoce una estimación inicial de la orientación del satélite utilizando el algoritmo TRIAD, para luego en las siguientes estimaciones utilizar el enfoque recursivo del EKF. Con estos estados ya conocidos, se utiliza el controlador PD para generar la acción de control que será aplicada y limitada por el tipo de magnetorquer a simular, generando un torque capaz de generar un cambio de orientación.

Para el caso de esta primera iteración, al no diseñar el EKF, se asume un error en el cuaternión para representar la estimación de la actitud del algoritmo de estimación, y aumentará o disminuirá en base a la calidad de los sensores elegidos según los niveles de componentes mencionados anteriormente, como se muestra en el esquema de la Figura 26, sin contar el ruido del giroscopio en este caso por razones mostradas en las siguientes secciones.

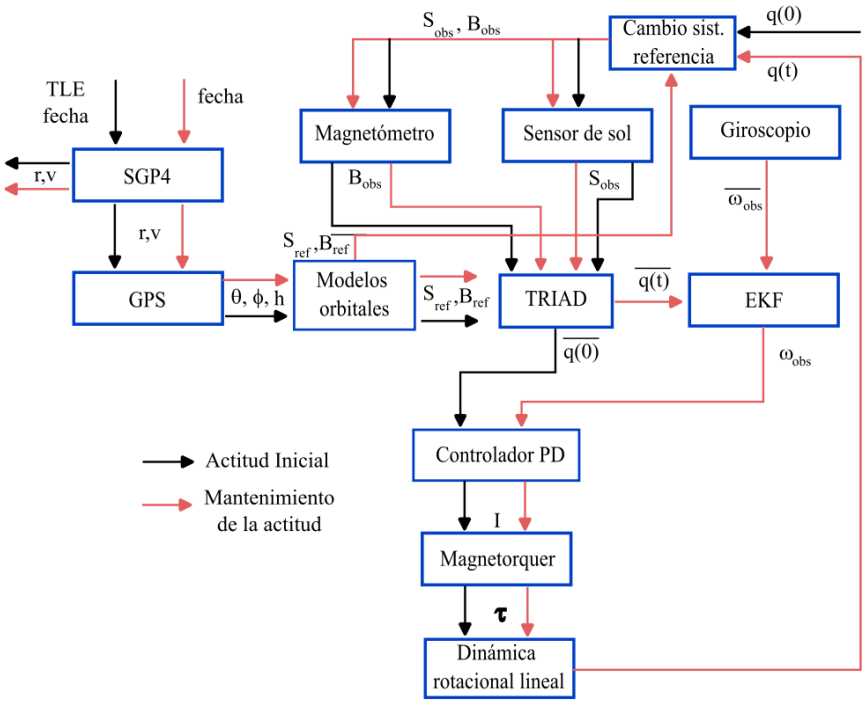


Figura 25. Diagrama de la suite de simulación ideal.

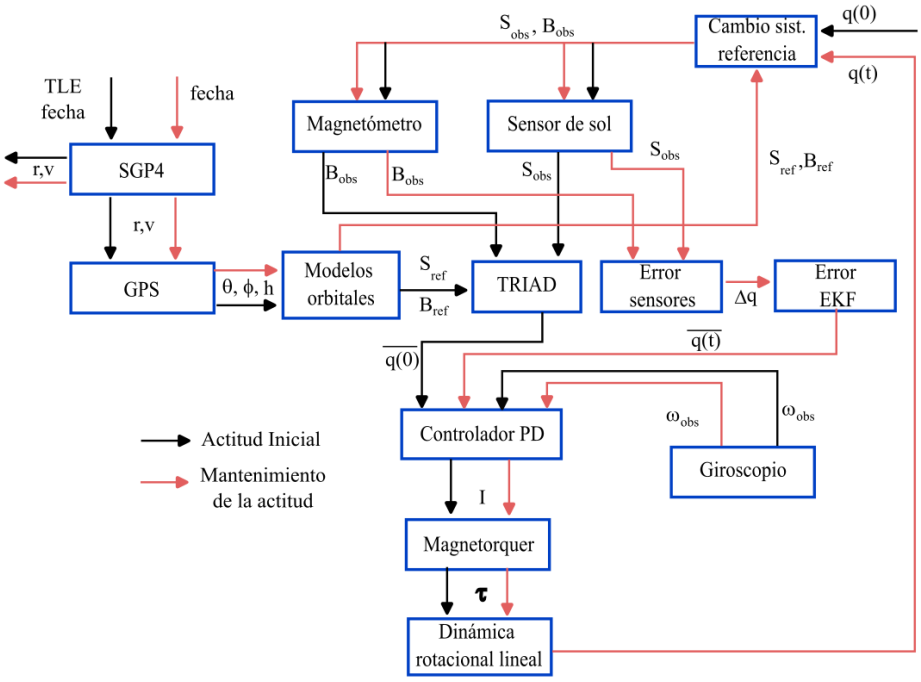


Figura 26. Diagrama de la primera iteración de la suite de simulación.

CAPÍTULO 6: Resultados y validación de la suite de simulación

En este capítulo se verán resultados en base a la elección de parámetros y datos de un *CubeSat* operativo, para obtener los MoP de apuntamiento con sus respectivos costos según los niveles de componentes físicos impuestos dentro de la simulación. Es relevante mencionar que se obtuvo la cuantificación de la exactitud de apuntamiento, el *jitter* y la agilidad, sin mostrar resultados para el *drift*, al no ser lo suficientemente relevante para las misiones de observación terrestre, y además al no simular de manera precisa y detallada las condiciones orbitales.

6.1 Condiciones y parámetros de simulación

Para obtener resultados de rendimiento, se utilizaron los datos de TLE del SUCHAI-3 con fecha de inicio del 01 de noviembre del 2023, que son las mismas condiciones iniciales utilizadas en el Capítulo 5 para visualizar el SGP4 y los modelos orbitales en ECI (IGRF y vector sol). Se debe tener en cuenta que estas condiciones, en conjunto con otros parámetros son dados para todos los niveles de sensores y actuadores impuestos dentro de la suite de simulación, y sirven para tener un parámetro comparativo entre los componentes físicos del ADCS.

En la Tabla 8 se muestran los valores de los parámetros recién mencionados utilizados en la simulación, que son basados en literatura [14, 44] o por elaboración propia. Cabe mencionar que q_i representa la orientación inicial real, con la cual se inicializa el marco de referencia del cuerpo respecto a ECI para representar el vector de observación inicial, aplicando después un cambio de sistema de referencia a LVLH. Esta condición inicial no es utilizada en la simulación, ya que para ello está la estimación del TRIAD. Por otro lado, la velocidad angular inicial ω_i si se utiliza como condición inicial en el simulador.

Tabla 8. Parámetros utilizados para la simulación [14].

$\boldsymbol{I_x}\ [\boldsymbol{kg \cdot m^2}]$	0.037
$\boldsymbol{I_y}\ [\boldsymbol{kg \cdot m^2}]$	0.036
$\boldsymbol{I_z}\ [\boldsymbol{kg \cdot m^2}]$	0.006
$[\boldsymbol{K_{p1}}, \boldsymbol{K_{p2}}, \boldsymbol{K_{p3}}]$	$[-11.71, 0.02154, -1.941]$
$[\boldsymbol{K_{d1}}, \boldsymbol{K_{d2}}, \boldsymbol{K_{d3}}]$	$[\ 2.231, -0.05916, -473.8]$
$\boldsymbol{\omega_{0,o}}\ \left[\frac{\boldsymbol{rad}}{\boldsymbol{s}}\right]$	0.00163
$\boldsymbol{T_{prop}}\ [\boldsymbol{s}]$	$400000\ [s]$
$\boldsymbol{q_i}\ [-]$	$\left[-\frac{0.7071}{\sqrt{3}}, \frac{0.7071}{\sqrt{3}}, -\frac{0.7071}{\sqrt{3}}, 0.7071\right]$
$\boldsymbol{\omega_i}\ \left[\frac{\boldsymbol{rad}}{\boldsymbol{s}}\right]$	[0.0001,0.0001,0.0001]

Es relevante mencionar que en el Anexo G se muestra un ejemplo a detalle sobre la obtención de los MoP de apuntamiento, siendo estos los pasos a seguir para los casos de estudio que se mencionarán en las siguientes secciones. Además, para el caso de los sensores, se considera el uso de la velocidad angular sin ruido, debido a que se presentaron inestabilidades al implementar el ruido blanco en base a la desviación estándar de cada giroscopio. Esto se debe principalmente a que la desviación estándar y el bias presenta valores mayores a las velocidades angulares obtenidas a través del tiempo, permitiendo que para trabajo futuro se estudie la asignación de pesos a las constantes derivativas para su inclusión dentro de la suite de simulación.

Además, se muestra a continuación la respuesta del sistema para componentes físicos de nivel alto al utilizar el TRIAD como el algoritmo de estimación de actitud en todo el periodo de simulación en la Figura 27, mostrándose como se menciona en la sección 5.4.1, que la utilización de este provoca una estabilización bastante alejada al *setpoint* (valor deseado) en el Roll debido a la mala estimación de la orientación del *CubeSat*, por lo que se comprueba que también afecta a los MoP de apuntamiento (en específico la exactitud de apuntamiento) el uso de diferentes calidad de algoritmos dentro de la suite de simulación.

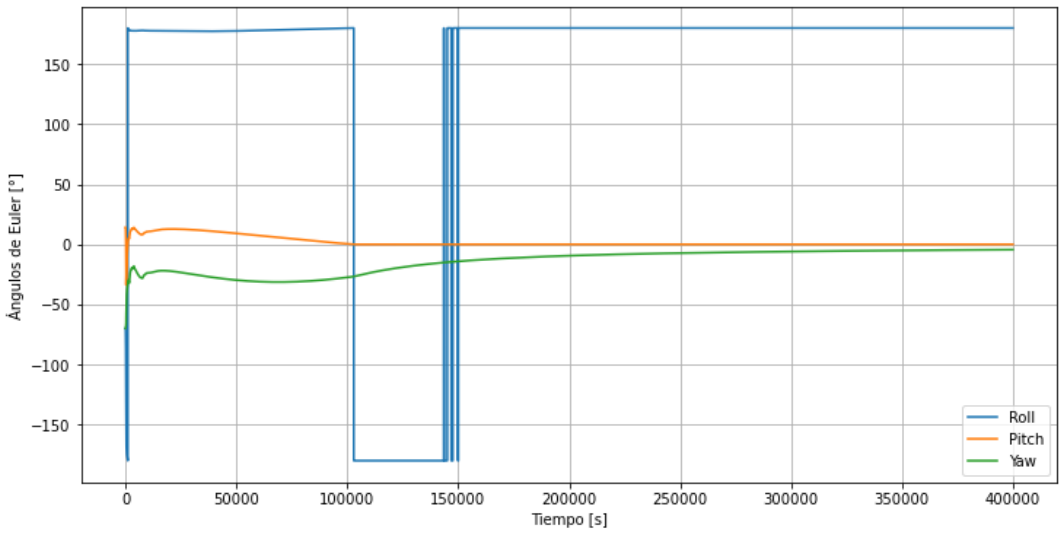


Figura 27. Ángulos de Euler entre marcos de referencia LVLH y cuerpo para sensores y actuadores de nivel alto.

6.2 Análisis de rendimiento vs costo según nivel de sensores

En esta sección se presentarán análisis respecto a los tres niveles de los sensores, utilizando el magnetorquer de nivel alto, para analizar cómo afecta a los MoP de apuntamiento la estimación de la actitud del satélite respecto a los componentes físicos y cuanto es el costo asociado.

En la Figura 28 se muestra el resultado de utilizar sensores de nivel bajo (magnetómetro y sensor de sol) con un buen actuador para estabilizar los ángulos de Euler, mostrándose altas fluctuaciones al intentar estabilizar el sistema tanto para el Roll como para el Pitch. También se observa que no se logra llegar al *setpoint* deseado en ninguno de los ángulos de Euler debido a que la acción de control ejercida en base a la estimación de la actitud es insuficiente respecto a la que se obtendría utilizando la orientación ideal. Por otro lado, se observa que tanto el Roll como el Yaw tienen un tiempo de

asentamiento similar y menor al del Roll. Para mejorar el rendimiento utilizando este tipo de sensores se podría mejorar el controlador al agregar una acción integradora al PD o simplemente utilizar un LQR.

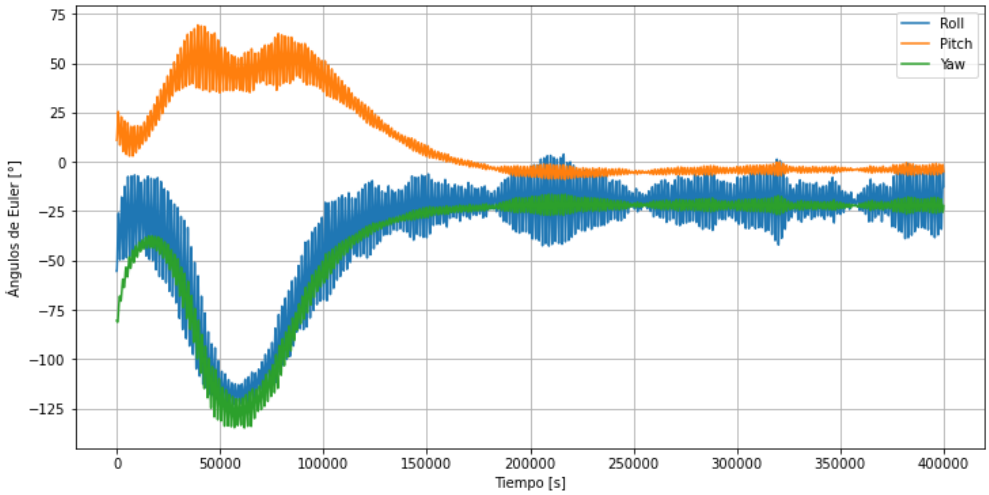


Figura 28. Ángulos de Euler entre marcos de referencia LVLH y del cuerpo para sensores de nivel bajo.

Para denotar la diferencia entre los tres niveles, se graficaron los casos de estudio para un ángulo de Euler, que para este caso fue el Yaw, y es representado en la Figura 29. De aquí se observa gráficamente que existe una mejor exactitud de apuntamiento para sensores de nivel alto, siendo alrededor de 9 [°], mientras que los niveles medio y bajo presentan un error absoluto de aproximadamente 16 [°] y 22 [°] respectivamente. Además, también de denotan mayores fluctuaciones en las vibraciones a medida que disminuye la calidad del sensor, mientras que el tiempo de asentamiento no presenta grandes variaciones a simple vista.

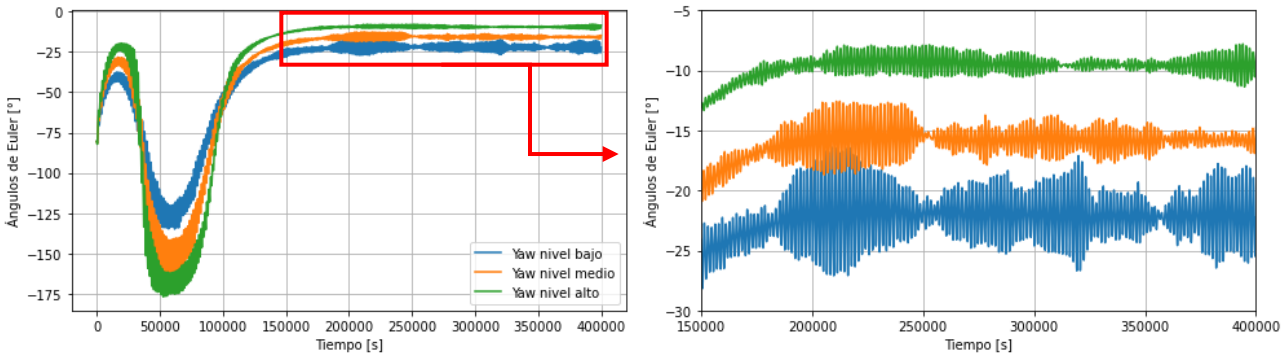


Figura 29. Comparación del Yaw para los niveles de sensores de manera gráfica

Teniendo los tres casos implementados, se deben analizar su costo en base a los SE *envelopes*. Se observa en la Tabla 9 el gasto por niveles de componente de potencia, masa y tamaño en volumen de los sensores de sol y magnetómetros, mostrándose que tal y como se espera, un sensor de nivel alto ocupa más espacio dentro del satélite, presenta mayor masa y tiene peaks de potencia mayores respecto a otros de menor calidad.

Tabla 9. Costo para la determinación de actitud del satélite.

Niveles (magnetómetro + sensor de sol)	Potencia máxima [W]	Masa[kg]	Volumen [mm^3]
Bajo	0.010+0	0.012+0.010	7 458+1 140
Medio	0.15+0	0.1+0.036	93 646+23 246
Alto	0.3+0.25	0.1+0.375	93 646+612 360

Además, se debe tener en cuenta la cuantificación del rendimiento en base a los resultados de los MoP de apuntamiento. Para ello, se muestra en la Tabla 10 la exactitud de apuntamiento, el *jitter* y la agilidad como norma de los valores obtenidos en cada ángulo de Euler, mostrándose que, a menores niveles de sensores, existe una menor exactitud de apuntamiento y una mayor densidad espectral de potencia, que se traduce a un mayor nivel energético utilizado en base a las vibraciones obtenidas en el filtro pasa alto. Se puede observar además que el tiempo de asentamiento varía levemente respecto al tipo de sensor utilizado, siendo estos componentes poco útiles para mejorar la agilidad del *CubeSat*.

Tabla 10. Rendimiento para la determinación de actitud del satélite en base a la norma de los MoP de apuntamiento.

Niveles	Exactitud de apuntamiento [°]	*Jitter* [W/Hz]	Agilidad [s]
Bajo	30.1	123	389 257
Medio	21.5	61.3	379 125
Alto	12.8	40.5	373 094

6.3 Análisis de rendimiento vs costo según nivel de actuadores

En esta sección se analizarán los niveles de los actuadores utilizando el sensor de nivel alto, para analizar cómo afecta a los MoP de apuntamiento la restricción del controlador en base a la calidad del magnetorquer.

En primera instancia se muestran los resultados de los ángulos de Euler para el magnetorquer de nivel bajo en la Figura 30. En ella se muestra un bajo nivel de fluctuaciones en las vibraciones a medida que el sistema se intenta estabilizar. Esto se debe principalmente a que, al tener como torque máximo un valor bajo, este genera frecuencias de menor nivel que provocan el nivel de vibraciones mostradas en la gráfica. Por otro lado, el sistema muestra un acercamiento al *setpoint* aun cuando se tiene un actuador de bajo nivel, mostrando la leve relevancia del actuador para la exactitud de apuntamiento. Finalmente, se observa un alto tiempo de asentamiento en los ángulos de Euler.

Además, se pueden observar en la Figura 31 la comparación para el Yaw entre los tres niveles de actuadores utilizados. Se observa que existe mayor relevancia respecto al tiempo de asentamiento, siendo mayor para un magnetorquer de peor calidad (y por lo tanto menos ágil) con un valor aproximado de 328 000 [s], mientras que para el nivel medio y alto existe un tiempo de asentamiento estimado de 280 000 [s] y 190 000 [s] respectivamente. Además, se puede observar que para el nivel

medio se muestran mayores amplitudes en las vibraciones presentes, sobre todo en los intervalos de tiempo entre 45 000 y 55 000 segundos, en las cuales se observa cierta dificultad con la estabilización del sistema, que es provocada por las aproximaciones del cuarto componente (otorgarle el valor 1 de manera restrictiva si la norma de los otros tres componentes del cuaternión es mayor a 1) en el modelo con los cuaterniones. Finalmente, no existe una diferencia notable en la exactitud de apuntamiento respecto al cambio de actuador utilizado.

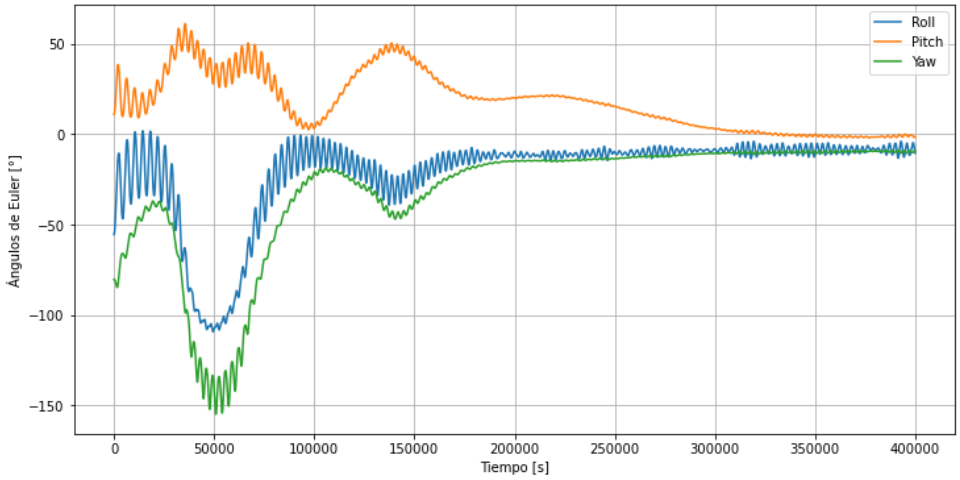


Figura 30. Ángulos de Euler entre marcos de referencia LVLH y del cuerpo para actuadores de nivel bajo.

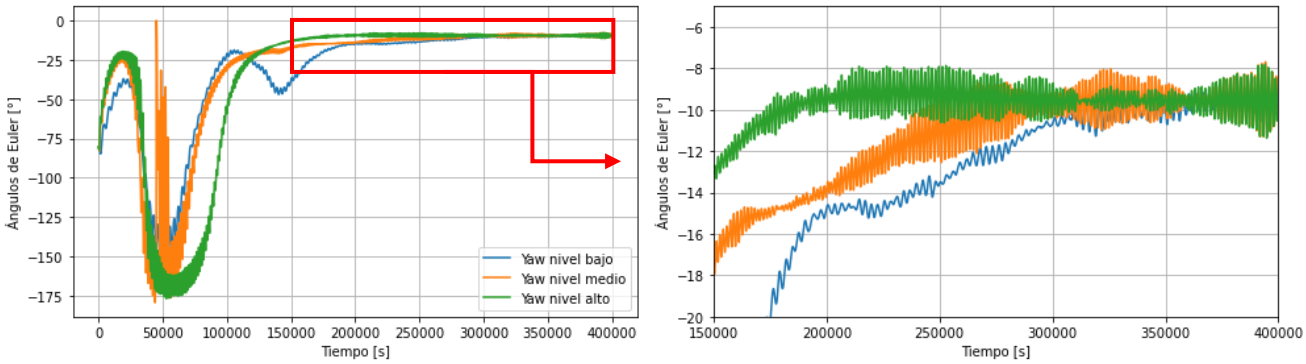


Figura 31. Comparación del Yaw para los niveles de actuadores de manera gráfica.

Al igual que en la sección anterior, se analizan los costos de los niveles de magnetorquers. Se observa en la

Tabla 11 el gasto por niveles de componente de potencia, masa y tamaño en volumen, mostrándose mayores gastos en componentes físicos de mayor calidad.

Tabla 11. Costo del actuador utilizado.

Niveles (magnetorquer)	**Potencia máxima [W]**	**Masa[kg]**	**Volumen [mm^3]**
Bajo	0.42	0.028	15 624
Medio	0.8	0.053	18 525
Alto	1.11	0.43	74 789

Por otro lado, se muestran en la Tabla 12 los valores cuantificados de los MoP de apuntamiento obtenido en la estabilización de la norma en los ángulos de Euler según el *setpoint* impuesto. Se presentan leves mejoras en la exactitud de apuntamiento a medida que se utilizan componentes de mayor calidad. Por otro lado, existe una mayor densidad espectral de potencia a medida que se utilizan actuadores con mayor momento dipolar máximo, debido a que, al ejercer torques de mayor magnitud, existen frecuencias mayores que causan un mayor *jitter*, generando un coste energético alto. Esto se observa de manera clara entre el nivel bajo y alto, pero no existe una relación con el nivel medio debido a la restricción del cuarto componente del cuaternión, que provoca más perturbaciones en el sistema. Finalmente, la agilidad es el MoP de apuntamiento más relevante respecto al uso de diferentes niveles del magnetorquer, al mostrarse variaciones significativas en el tiempo de asentamiento, debido al aumento de la acción de control máxima permitida por los actuadores de alto nivel.

Tabla 12. Rendimiento respecto al actuador utilizado.

Niveles	**Exactitud de apuntamiento [°]**	***Jitter* [W/Hz]**	**Agilidad [s]**
Bajo	13.4	10.4	566 288
Medio	13.2	84.3	485 209
Alto	12.8	40.5	373 094

6.4 Análisis de cambios combinados

Para una mejor visualización de los resultados se emplea un análisis de cambios combinados, en el cual se analizará los niveles de los componentes físicos de tal manera que se muestre mediante una matriz de riesgo como afecta la utilización de un tipo de sensores y actuadores a un MoP de apuntamiento seleccionado. En este trabajo se muestra el análisis de solo dos índices de rendimiento, basados en los resultados obtenidos en las subsecciones anteriores y en el tiempo disponible para su realización.

Antes de emplear las matrices de riesgo, se muestra en la Tabla 13 las representaciones de los códigos de colores y de números utilizados, basándose en la mejor/peor combinación de sensor y actuador, siendo la mejor opción obtenida mediante el numero “1” con un color verde claro, mientras que la peor combinación a utilizar para el MoP de apuntamiento esta denotada con un color rojo y el número 6, con sus respectivas definiciones intermedias.

Ya sabida la representación de los códigos de colores y números, se emplea la matriz de riesgo en primera instancia para la exactitud de apuntamiento en la Tabla 14. Se puede observar que el cambio en la calidad del sensor afecta en gran medida a una exactitud de apuntamiento más cercana al *setpoint*, mientras que la utilización de un nivel de actuador más alto presenta mejorías leves para este MoP de apuntamiento. Por otro lado, se realiza otra matriz de riesgo para la agilidad en la Tabla 15, presentando una mayor relevancia la calidad del actuador para la obtención de un *CubeSat* con mayor agilidad, mientras que el uso de niveles de sensores alto disminuye de manera ínfima el tiempo de asentamiento.

Tabla 13. Representación de la matriz de riesgo en el análisis de cambios combinados por MoP de apuntamiento.

Alto rendimiento	1: Mejor combinación según los componentes físicos impuestos en la simulación.
	2: Es una buena opción respecto al MoP de apuntamiento analizado.
Rendimiento medio	3: Presenta una combinación aceptable de sensores y actuadores para el MoP de apuntamiento analizado.
	4: Demuestra una opción regular respecto a la utilización de sus componentes físicos.
Bajo rendimiento	5: Presenta un rendimiento ineficiente en comparación con las otras opciones presentadas.
	6: Peor opción para elegir respecto al MoP de apuntamiento analizado.

Tabla 14. Matriz de riesgo respecto a los niveles de sensores y actuadores para la exactitud de apuntamiento.

		Nivel del sensor		
		Bajo	Medio	Alto
Nivel del actuador	Bajo	6	4	2
	Medio	5	4	2
	Alto	5	3	1

Tabla 15. Matriz de riesgo respecto a los niveles de sensores y actuadores para la agilidad.

		Nivel del sensor		
		Bajo	Medio	Alto
Nivel del actuador	Bajo	6	5	5
	Medio	4	4	3
	Alto	2	2	1

6.5 Costo del proyecto

Al llevarse a cabo un proyecto utilizando solo recursos computacionales y software de libre uso, el costo del proyecto solo se cuantifica en las horas hombre trabajadas para su realización y el costo de mantención del equipo con el cual se realizó la suite de simulación, sabiendo que la realización del proyecto tomó 4 meses para su finalización, cuyo resumen se encuentra en la Tabla 16.

Tabla 16. Horas hombre y costo del equipo utilizado para la realización de la suite de simulación

Activo	Inversión [CLP]	Costo fijo [CLP]	Monto total [CLP]
Sueldo Ingeniero	-	$1 200 000	$4 800 000
Mantención computador	$100 000	-	$100 000
			$4 900 000

CAPÍTULO 7: Conclusiones

Durante la ejecución del proyecto, se logró caracterizar y cuantificar los componentes del ADCS en niveles bajo, medio y alto en comparación con componentes de *CubeSat* disponibles comercialmente (COTS). Esta evaluación se respaldó mediante la recopilación anual de información sobre componentes utilizados en *Small Satellites* de la NASA, que proporcionó nombres, proveedores y costos asociados. La selección se basó principalmente en un alto *Technology Readiness Level* (TRL), demostrando su eficacia en misiones espaciales, y en la coherencia con la premisa de que un mayor costo respecto a la masa, potencia, tamaño y precio conlleva a mejores características sobre el rendimiento del apuntamiento del satélite.

Posteriormente, se logró caracterizar cuantitativamente los cuatro MoP de apuntamiento. Este logro se basó en investigaciones previas, obteniendo ideas de artículos relacionados con el ADCS para comprender el funcionamiento de la exactitud de apuntamiento, *jitter*, *drift* y agilidad. La exactitud de apuntamiento se cuantificó mediante el error absoluto de apuntamiento entre el eje de rotación requerido y el obtenido en grados. El *jitter* se midió según la densidad espectral de potencia obtenida del filtro pasa alto ejercido sobre la respuesta del sistema, que representa la cantidad de energía gastada por las vibraciones. El *drift* se representó como una velocidad angular mínima para prevenir la caída del satélite, y finalmente la agilidad se evaluó según el tiempo de asentamiento necesario para estabilizar el sistema con una determinada banda de asentamiento.

Durante la recopilación de información para cuantificar los MoP de apuntamiento, también se obtuvo información sobre la relación entre el hardware y software del ADCS y el rendimiento de apuntamiento. Esto permitió establecer cómo los sensores, algoritmos de determinación de actitud, controladores y actuadores afectan cada MoP de apuntamiento, determinando la influencia de estos componentes en dichos parámetros de manera teórica.

En la etapa inicial del desarrollo del simulador, se identificaron herramientas y modelos útiles para la implementación de la dinámica orbital y de actitud. Para la dinámica orbital, se seleccionó el propagador SGP4, capaz de obtener la posición y velocidad del satélite considerando perturbaciones a bajas alturas de la Tierra. Por otro lado, se encontraron modelos capaces de estimar la orientación del satélite y así encontrar su dinámica rotacional dentro la suite de simulación creada. También se encontraron dentro del estado del arte simuladores similares capaces de modelar ambas dinámicas del satélite, que no se pueden obtener debido a su alto precio, pero que sirve como referencia para la realización del proyecto.

Posteriormente, se avanzó significativamente en el diseño de la arquitectura del simulador capaz de estimar el rendimiento y el costo de apuntamiento. Los modelos de dinámica orbital y de actitud fueron implementados con éxito en Python. Se logró simular las mediciones de los sensores solares y magnetómetros, y se implementó un algoritmo de determinación de actitud determinista como el TRIAD. No obstante, quedó pendiente la implementación de la respuesta recursiva del filtro extendido de Kalman, representándolo solo como un error de estimación en la actitud del satélite, y también faltó la inclusión del ruido del giroscopio en la velocidad angular sin que el sistema de inestabilice. También se modeló el controlador proporcional-derivativo (PD) para lograr el *setpoint* entre los

marcos de referencia del *CubeSat*, generando la acción de control con los magnetorquers dentro del simulador de manera exitosa utilizando un modelo lineal.

Con estos avances, se estableció una relación costo-rendimiento en relación la exactitud de apuntamiento, el *jitter* y la agilidad según los tipos de componentes físicos seleccionados (no se cuantificó el *drift* en este trabajo al no simular de manera detallada la dinámica orbital del sistema), utilizando los datos del SUCHAI-3 para dicha validación. Se reconoció que existe una mayor relevancia en la selección de sensores para obtener una mayor exactitud de apuntamiento de los ángulos de Euler, mientras que para los actuadores se relacionan de manera significativa con la agilidad. También se observó que mayor calidad en los sensores genera menor *jitter*, mientras que magnetorquers con mayor capacidad de momento dipolar (acción de control) genera mayores vibraciones, y por lo tanto un *jitter* mayor. Por lo tanto, se confirma que un mayor rendimiento en el apuntamiento exige de un mayor costo en base a la calidad de los componentes físicos. También se confirma la influencia de la calidad del algoritmo de determinación de actitud, ya que al utilizar solo el TRIAD a través del tiempo, al ser un algoritmo simple y poco exacto, provocó que las acciones de control no logren estabilizar el sistema en el valor deseado.

Trabajos Futuros

Para el desarrollo de una suite de simulación realista, basada en las mediciones de componentes físicos como lo son en este caso los magnetómetros, los sensores de sol y los giroscopios, es necesario implementar el filtro de Kalman extendido (EKF) como algoritmo de estimación de actitud. Se busca que este implementado de tal manera que exista el mínimo error proveniente del algoritmo para la determinación de actitud del *CubeSat*, por lo que es necesario probar distintos modelos de sensor y utilizar el más adecuado, basado siempre en el realismo de las mediciones y del ruido inherente a los componentes y no a un error aleatorio impuestos en base a niveles.

Además, se debe implementar el ruido del giroscopio sin que existan inestabilidades en el control de la orientación del satélite. Para ello se debe aplicar una asignación de pesos en las constantes derivativas del controlador para incluir el ruido del giroscopio dentro del simulador o quedarse con la premisa de utilizar el ruido blanco proveniente solo de los otros sensores simulados.

Por otro lado, también es necesario diseñar acciones de control basadas en otro actuador como la rueda de reacción, que tiene capacidad de generar mayores torques, con el objetivo de comparar y reconocer en qué casos es mejor utilizar uno u el otro. Para ello, se debe hacer un análisis sobre el modelamiento de la rueda de reacción e implementarlo dentro de la suite de simulación, sabiendo que se está basando en un sistema linealizado. Además, es relevante tener en cuenta el precio dentro de los costos, preguntando a los proveedores de los componentes físicos elegidos como niveles en la suite de simulación.

Finalmente, es necesario que, una vez terminada esta versión del simulador, esta se reestructure basándose en la optimización de los componentes físicos. Esto se hará eligiendo un optimizador no lineal que sea capaz de entregar como respuesta el nivel de componentes físicos (sensor y actuador) que logre un determinado MoP de apuntamiento y/o costo, el cual será entregado como entrada en conjunto con los parámetros ya discutidos del *CubeSat* como la geometría, fechas, etc. Esto debido a

que generalmente las restricciones impuestas en las misiones con *CubeSat* son los MoP de apuntamiento y/o costo, haciendo más fácil seleccionar al usuario el tipo de componente físico que necesita en su misión. Teniendo esto en cuenta, se debe reordenar el código base y presentar una interfaz amigable al usuario, de manera tal que entregue las entradas a la suite de simulación y pueda obtener los resultados deseados.

Referencias

[1] nanosats.eu, «Nanosats Database,» 2023. [En línea]. Available: https://www.nanosats.eu/.

[2] SatCatalog, «CubeSat Launch Costs,» 2022. [En línea]. Available: https://www.satcatalog.com/insights/cubesat-launch-costs/#:~:text=Rocket%20Lab,-In%20general%2C%20Rocket&text=The%20company%20offers%20a%20range,launch%20to%20LEO%20or%20SSO..

[3] nanosats, «CubeSats Costs,» 2023. [En línea]. Available: https://www.nanosats.eu/tables#keywords.

[4] B. Wang, «NextBigFuture,» 10 diciembre 2020. [En línea]. Available: https://www.nextbigfuture.com/2019/12/spacex-starlink-satellites-cost-well-below-500000-each-and-falcon-9-launches-less-than-30-million.html.

[5] ESA, «HOW IT COST?,» 2023. [En línea]. Available: https://www.esa.int/Science_Exploration/Human_and_Robotic_Exploration/International_Space_Station/How_much_does_it_cost.

[6] N. T. Tillman, «Space,» 30 Enero 2022. [En línea]. Available: https://www.space.com/15892-hubble-space-telescope.html.

[7] «mapscaning,» 2 Marzo 2023. [En línea]. Available: https://mapscaping.com/optical-satellites/#:~:text=Optical%20sensors%20on%20Earth%20observation,is%20reflected%20back%20toward%20space..

[8] W. J. Larson y J. R. Wertz, Space Mission Analysis Design, Space technology library, 1999.

[9] M. Tacul, «IDENTIFICACIÓN Y CARACTERIZACIÓN DE ÍNDICES DE PERFORMANCE EN EL APUNTAMIENTO DE PAYLOAD EN CUBESATS,» Concepción, 2023.

[10] J. P. Mason, M. Baumgart, B. Rogler, C. Downs, M. Williams, T. N. Woods, S. Palo, P. C. Chamberlin, S. Solomon, A. Jones, X. Li, R. Kohnert y A. Caspi, «MinXSS-1 CubeSat On-Orbit Pointing and Power Performance: The First Flight of the Blue Canyon Technologies XACT 3-axis Attitude Determination and Control System,» *Journal of Small Satellites,* vol. 6, pp. 651-662, 2017.

[11] S.-G. Kim y J.-S. Chae, «Attitude Control System Performance Estimation for Next Generation Small Satellite 1,» *World congress on Aeronautics, Nano, Bio, Robotics and Energy,* 2015.

[12] R. Votel y D. Sinclair, «Comparison of Control Moment Gyros and Reaction Wheels for Small Earth-Observing Satellites,» de *Small Satellite Conference*, 2012.

[13] E. Babcock y T. Bretl, «CubeSat Attitude determination via Kalman Filtering of Magnetometer and Solar Cell data,» de *Conference on Small Satellites*, Illinois, 2011.

[14] D. M. Torczynski, . R. Amini y P. Massioni, «Magnetorquer Based Attitude Control for a Nanosatellite Testplatform,» de *AIAA Infotech@Aerospace 2010*, 2010.

[15] D. Velez, E. Dawson y N. Nassif, «Attitude Determination and Control Subsystem design for a CubeSat,» Digital WPI, Massachussetts, 2012.

[16] A. Annenkova, S. Biktimirov, K. Latyshev, A. Mahfouz, P. Mukhachev y D. Prytikin, «Cubesat ADCS model for preliminary design procedures within a concurrent design approach,» de *AIP Conference Proceedings*, 2019.

[17] A. Rassõlkina, T. Vaimanna, P. Orgb, A. Leibakc, R. Gordond y E. Priidel, «ADCS development for student CubeSat satellites – TalTech case study,» *Proceedings of the Estonian Academy of Sciences,* vol. 70, nº 3, p. 268–285, 2021.

[18] Princeton Satellite System, «Spacecraft Control Toolbox,» 2017. [En línea]. Available: http://support.psatellite.com//sct/index.php. [Último acceso: 09 Octubre 2023].

[19] MathWorks, «Model and Simulate CubeSats,» [En línea]. Available: https://la.mathworks.com/help/aeroblks/model-and-simulate-cubesats.html. [Último acceso: 17 octubre 2023].

[20] Valispace, «Fan tutorials,» [En línea]. Available: https://docs.valispace.com/vhd/fan-tutorials. [Último acceso: 13 Octubre 2023].

[21] NASA, «State-of-the-Art of Small Spacecraft Technology,» Small Spacecraft Systems Virtual Institute , California, 2023.

[22] A. Maricahuin, «Diseño de un simulador solar para la determinación de actitud de Cubesats en LEO basado en un sistema de iluminación controlado y orientable,» Universidad de Concepción, Concepción, 2023.

[23] A. Chavez Jiménez, «On the Coupling of Orbit and Attitude Determination of Satellite Formations from Atmospheric Drag,» TUDelft, Delft, 2020.

[24] H. Curtis, Orbital Mechanics for Engineering Students, Florida: Elsevier, 2014.

[25] J. Wertz, Spacecraft Attitude Determination and Control, Dordrecht: D.Reidel, 1984.

[26] AcademiaLab, «Elementos orbitales,» [En línea]. Available: https://academia-lab.com/enciclopedia/elementos-orbitales/. [Último acceso: 02 octubre 2023].

[27] F. Landis Markley y J. Crassidis, Fundamentals of Spacecraft Attitude Determination and Control, Springer, 2014.

[28] J. J. Sellers, «Chapter 3: Space Environment,» de *Understanding Space: An Introduction to* , Space Technology Series, 2004, pp. 71-90.

[29] J. Junkins y H. Scraub, Analytical Mechanics of Aerospace Systems, AIAA, 2009.

[30] T. Kuwahara, «Introduction to CubeSat Attitude Control System,» Sendai, 2021.

[31] R. Tedrake, «Linear Quadratic Regulators,» [En línea]. Available: https://underactuated.mit.edu/lqr.html. [Último acceso: 3 12 2023].

[32] Ansys, «Systems Tool Kit,» Ansys corp, [En línea]. Available: https://www.ansys.com/products/missions/ansys-stk#tabs2-5797c1dc80-content-1. [Último acceso: 09 Octubre 2023].

[33] Valispace, «Valispace,» [En línea]. Available: https://www.valispace.com/. [Último acceso: 13 Octubre 2023].

[34] C. Dennehy y O. Álvarez-Salazar, «A SURVEY OF THE SPACECRAFT LINE-OF-SIGHT JITTER PROBLEM,» NASA, 2020.

[35] Z. Ye, Y. Xu, S. Zheng,, X. Tong, X. Xu, S. Liu, H. Xie, S. Liu, C. Wei y U. Stilla, «Resolving time-varying attitude jitter of an optical remote sensing satellite based on a time-frequency analysis,» *Opt Express*, nº 28, pp. 15805-15823, 2020.

[36] G. Lavezzi, M. Eivind Grøtte y M. Ciarcià, «Attitude Control Strategies for an Imaging CubeSat,» de *Research Gate*, 2019.

[37] D. Vallado y P. Crawford, «SGP4 Orbit Determination,» *AIAA*, 2008.

[38] a.i solutions, «FreeFlyer Software,» 2023. [En línea]. Available: https://ai-solutions.com/freeflyer-astrodynamic-software/.

[39] P. Alken, E. Thébault, C. Beggan, y et al., «International Geomagnetic Reference Field: the thirteenth generation,» *Earth, Planets, Space*, vol. 73, nº 49, 2021.

[40] National Centers for Enviromental Information, «International Geomagnetic Reference Field (IGRF),» 2021. [En línea]. Available: https://www.ncei.noaa.gov/products/international-geomagnetic-reference-field. [Último acceso: 29 11 2023].

[41] D. Guichard, Calculus: Early Transcendentals, Lyrix, 2017.

[42] A. Bacciotti, Stability and Control of Linear Systems, Springer, 2019.

[43] K. Fuglsang Jensen y K. Vinther, «Attitude Determination and Control System for AAUSAT3,» Aalborg University, Aalborg, 2010.

[44] P. Baranwal, B. Karabee y T. Kaushik, «Comparative Study of Classical and Fuzzy PID Attitude Control System with Extended Kalman Filter Feedback for Nanosatellites.,» de *69th International Astronautical Congress*, Bremen, 2018.

[45] Silicon Sensing Systems, «Giroscopios CRH03,» [En línea]. Available: https://www.siliconsensing.com/products/gyroscopes/crh03/. [Último acceso: 16 Octubre 2023].

[46] NewSpace Systems, «Giroscopio NSGY-001,» [En línea]. Available: https://satcatalog.s3.amazonaws.com/components/326/SatCatalog_-_NewSpace_Systems_-_NSGY-001_-_Datasheet.pdf?lastmod=20210708044339. [Último acceso: 2023 11 30].

[47] AAC Clyde Space, «Magnetometro MM200,» [En línea]. Available: https://www.aac-clyde.space/what-we-do/space-products-components/adcs/mm200. [Último acceso: 16 Octubre 2023].

[48] AAC CLYDE SPACE, «Magnetómetro MAG-3,» [En línea]. Available: https://www.aac-clyde.space/wp-content/uploads/2021/11/MAG%C2%AD3.pdf. [Último acceso: 30 11 2023].

[49] Space Micro, «Sun sensors CSS-01, CSS-02,» [En línea]. Available: https://www.satcatalog.com/component/css-01-02/. [Último acceso: 16 Octubre 2023].

[50] Space Micro, «Sun Sensor MSS-01,» [En línea]. Available: https://www.satcatalog.com/component/mss-01-02/. [Último acceso: 16 Octubre 2023].

[51] Bradford Space, «Sun Sensor FSS,» [En línea]. Available: https://satsearch.co/products/bradford-fine-sun-sensor. [Último acceso: 16 Octubre 2023].

[52] CubeSpace, «CubeWheel y CubeTorquer small,» [En línea]. Available: https://www.cubespace.co.za/downloads/gen_1_sensors_actuators_jan_2023_web.pdf. [Último acceso: 16 Octubre 2023].

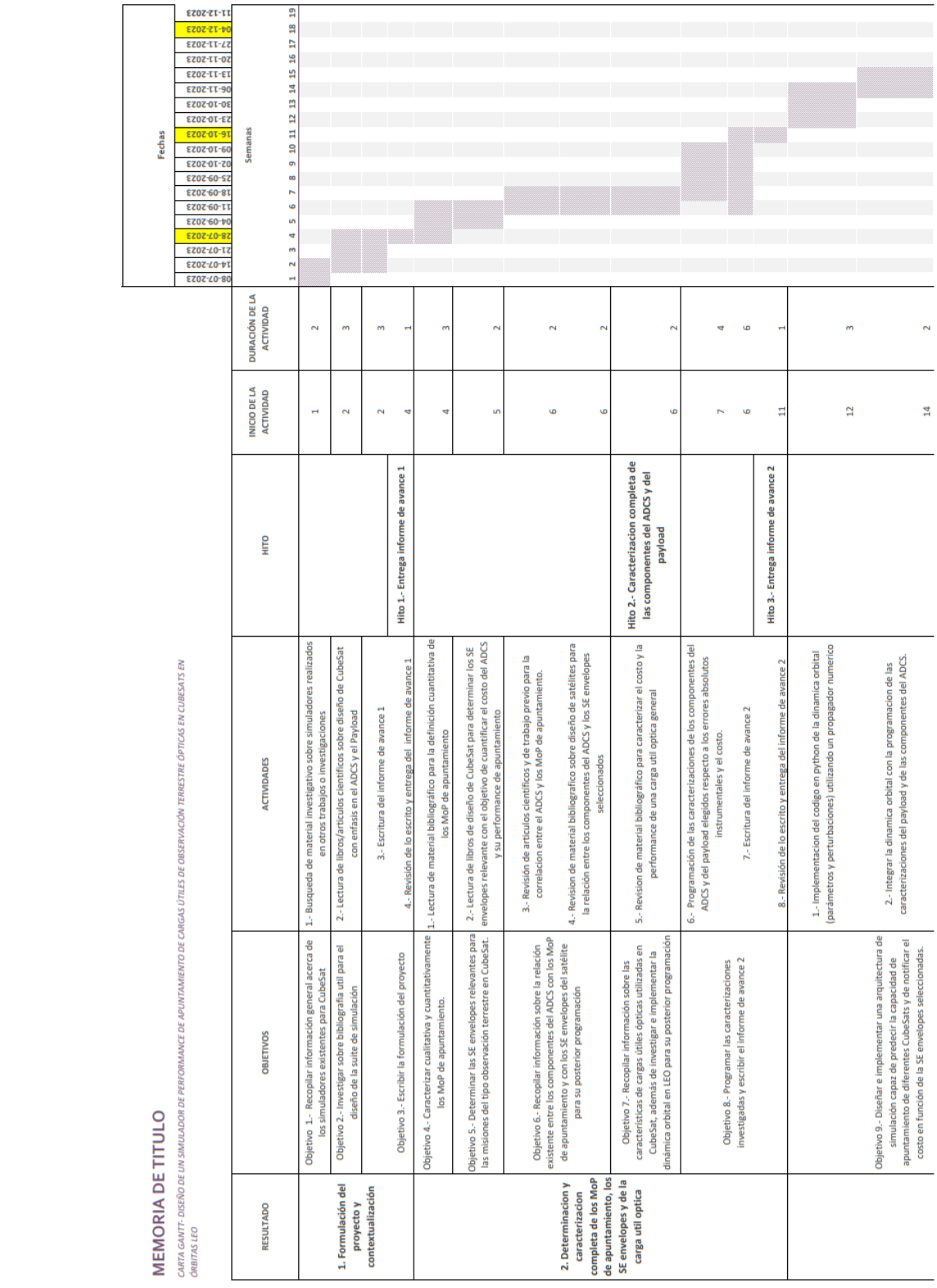
[53] RocketLab, «Rueda de reacción RW4-1.0,» [En línea]. Available: https://www.rocketlabusa.com/assets/Uploads/1-NMS-product-sheet.pdf. [Último acceso: 16 Octubre 2023].

[54] Blue Canyon Technologies, «Rueda de reaccion RW4,» [En línea]. Available: https://storage.googleapis.com/blue-canyon-tech-news/1/2023/04/ReactionWheels.pdf. [Último acceso: 16 Octubre 2023].

[55] NewSpace systems, «Magnetorquer NCTR-M012,» [En línea]. Available: https://www.newspacesystems.com/wp-content/uploads/2021/10/NewSpace-Magnetorquer-Rod_20211018a.pdf. [Último acceso: 16 Octubre 2023].

[56] ZARM Technik, «Magnetorquer MT15-1,» [En línea]. Available: https://satcatalog.s3.amazonaws.com/components/129/SatCatalog_-_ZARM_Technik_AG_-_MT15-1_-_Datasheet.pdf?lastmod=20210708025105. [Último acceso: 16 Octubre 2023].

Anexo A: Carta Gantt



MEMORIA DE TITULO

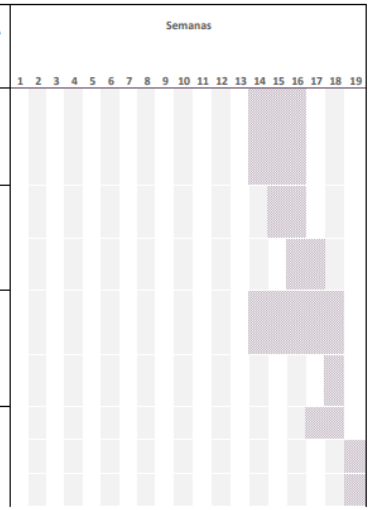
CARTA GANTT- DISEÑO DE UN SIMULADOR DE PERFORMANCE DE APUNTAMIENTO DE CARGAS UTILES DE OBSERVACIÓN TERRESTRE ÓPTICAS EN CUBESATS EN ÓRBITAS LEO

RESULTADO	OBJETIVOS	ACTIVIDADES	HITO	INICIO DE LA ACTIVIDAD	DURACIÓN DE LA ACTIVIDAD
1. Formulación del proyecto y contextualización	Objetivo 1.- Recopilar información general acerca de los simuladores existentes para CubeSat	1.- Busqueda de material investigativo sobre simuladores realizados en otros trabajos o investigaciones		1	2
	Objetivo 2.- investigar sobre bibliografia util para el diseño de la suite de simulación	2.- Lectura de libros/articulos cientificos sobre diseño de CubeSat con enfasis en el ADCS y el Payload		2	3
	Objetivo 3.- Escribir la formulación del proyecto	3.- Escritura del informe de avance 1		2	3
		4.- Revisión de lo escrito y entrega del informe de avance 1	Hito 1.- Entrega informe de avance 1	4	1
2. Determinacion y caracterizacion completa de los MoP de apuntamiento, los SE envelopes y de la carga util optica	Objetivo 4.- Caracterizar cualitativa y cuantitativamente los MoP de apuntamiento.	1.- Lectura de material bibliográfico para la definición cuantitativa de los MoP de apuntamiento		4	3
	Objetivo 5.- Determinar las SE envelopes relevantes para las misiones del tipo observación terrestre en CubeSat.	2.- Lectura de libros de diseño de CubeSat para determinar los SE envelopes relevante con el objetivo de cuantificar el costo del ADCS y su performance de apuntamiento		5	2
	Objetivo 6.- Recopilar información sobre la relación existente entre los componentes del ADCS con los MoP de apuntamiento y con los SE envelopes del satélite para su posterior programación	3.- Revisión de articulos cientificos y de trabajo previo para la correlacion entre el ADCS y los MoP de apuntamiento.		6	2
		4.- Revision de material bibliografico sobre diseño de satélites para la relación entre los componentes del ADCS y los SE envelopes seleccionados		6	2
	Objetivo 7.- Recopilar información sobre las características de cargas útiles ópticas utilizadas en CubeSat, además de investigar e implementar la dinámica orbital en LEO para su posterior programación	5.- Revision de material bibliográfico para caracterizar el costo y la performance de una carga util optica general	Hito 2.- Caracterizacion completa de las componentes del ADCS y del payload	6	2
	Objetivo 8.- Programar las caracterizaciones investigadas y escribir el informe de avance 2	6.- Programación de las caracterizaciones de los componentes del ADCS y del payload elegidos respecto a los errores absolutos instrumentales y el costo.		7	4
		7.- Escritura del informe de avance 2		6	6
		8.- Revisión de lo escrito y entrega del informe de avance 2	Hito 3.- Entrega informe de avance 2	11	1
	Objetivo 9.- Diseñar e implementar una arquitectura de simulación capaz de predecir la capacidad de apuntamiento de diferentes CubeSats y de notificar el costo en función de la SE envelopes seleccionadas.	1.- Implementacion del codigo en python de la dinamica orbital (parámetros y perturbaciones) utilizando un propagador numerico		12	3
		2.- Integrar la dinamica orbital con la programacion de las caracterizaciones del payload y de las componentes del ADCS.		14	2

Fechas: 08-07-2023, 14-07-2023, 21-07-2023, 28-07-2023, 04-09-2023, 11-09-2023, 18-09-2023, 25-09-2023, 02-10-2023, 09-10-2023, 16-10-2023, 23-10-2023, 30-10-2023, 06-11-2023, 13-11-2023, 20-11-2023, 27-11-2023, 04-12-2023, 11-12-2023

Semanas: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

RESULTADO	OBJETIVOS	ACTIVIDADES	HITO	INICIO DE LA ACTIVIDAD	DURACIÓN DE LA ACTIVIDAD
3. Diseño y Validacion de la arquitectura de simulacion		3.- Adicionar al simulador con una interfaz intuitiva que permita a los usuarios seleccionar los parámetros orbitales, componentes del ADCS y caracteristicas del payload	**Hito 4.- Finalizacion del simulador**	14	3
	Objetivo 10.- Validar los resultados del simulador en base a datos de la literatura de al menos una misión exitosa.	4.- Investigar y seleccionar al menos una misión espacial exitosa con características similares que sea relevante para el simulador.		15	2
		5.- Comparar los resultados obtenidos por el simulador con los datos reales de la misión	**Hito 5.- Validación del simulador**	16	2
	Objetivo 11.- Escribir el informe final	6.- Escritura del informe final		14	5
		7.- Revisión de los escrito y entrega del informe final	**Hito 6.- Entrega Informe final**	18	1
4. Defensa de la memoria de titulo	Objetivo 12.- Preparar la presentación en el power point	1.- Elaborar una presentación en ppt		17	2
	Objetivo 13.- Ensayar la presentación	2.- Presentar ante profesor guia y equipo		19	1
	Objetivo 14.- Entregar la presentación y defender el proyecto	3.- Revisión y entrega del ppt y su posterior defensa	**Hito 7. Presentación final**	19	1



Anexo B: Torques externos debido a las perturbaciones

Gradiente de gravedad: La órbita alrededor de la Tierra es posible gracias a la caída libre, que consiste en que la fuerza gravitacional hace caer al satélite constantemente sobre él. Sin embargo, debido a la velocidad horizontal que presenta la nave espacial en órbita, se mueve lo suficientemente rápido como para que su trayectoria curva coincida con la curvatura de la Tierra.

Si bien esta es la perturbación clave para que funcionen la orbitas en los satélites, también presenta consecuencias negativas para estos. Una de ellas es que la variación en el campo gravitacional de la Tierra a lo largo del *CubeSat* puede generar un gradiente gravitacional, causando un torque que se debe considerar al momento de analizar la dinámica de actitud del satélite. La ecuación que representa el torque se puede representar de la siguiente manera según [8]:

$$T_g = \frac{3\mu}{2\|R\|^3}\left(\frac{\boldsymbol{R}}{\|R\|}\times\begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix}\frac{\boldsymbol{R}}{\|R\|}\right) \approx 6{\omega_{0,o}}^2\begin{bmatrix} (I_z - I_y)\delta q_0 \\ (I_z - I_x)\delta q_1 \\ 0 \end{bmatrix}$$

Donde:

T_g : Es el torque de gravedad máximo

μ : Es la contante de gravedad de la Tierra $\left(3.988\ x\ 10^{14}\ \left[\frac{m^3}{s^2}\right]\right)$

$\boldsymbol{R}$: Posición de la órbita desde el centro de la tierra

I_x, I_y y I_z : Momentos de inercia en los ejes x, y, z respectivamente $[kg * m^2]$

Torque debido al arrastre atmosférico: La fuerza de arrastre en orbitas bajas generan torques externo que pueden afectar a la dinámica de actitud. Esta tiene una ecuación la cual se representa a continuación [8]:

$$T_a = F(c_{pa} - cg) = FL$$
$$F = \frac{1}{2}(\rho c_d A V^2)$$

Donde:

T_a : Torque ejercido por la fuerza de arrastre $[Nm]$

L : Offset del centro de masa respecto al centro de presión $[m]$

F : Fuerza de arrastre $[N]$

ρ : Densidad atmosférica $\left[\frac{kg}{m^3}\right]$

c_d : Coeficiente de arrastre [-]

A : Superficie enfrentada a la fuerza de arrastre $[m^2]$

V : Velocidad del satélite $\left[\frac{m}{s}\right]$

Presión debido a la radiación solar [8]: Esta depende en gran medida del tipo de superficie que está siendo iluminada. Una superficie puede ser transparente, absorbente o reflectante, pero la mayoría de las superficies son una combinación de las tres. Los reflectores se clasifican como difusos o especulares. En general, los conjuntos solares son absorbentes y el cuerpo de la nave espacial es un reflector. El peor caso de par de torsión debido a la radiación solar es:

$$T_{sp} = F(c_{ps} - cg)$$

$$F = \frac{F_s}{c} A_s(1+q)\cos(i)$$

Donde:

F_s : Constante solar (1.367 $\left[\frac{W}{m^2}\right]$)

c : Velocidad de la luz ($3x10^{18}$ $\left[\frac{m}{s}\right]$)

A_s : Area de la superficie

q : Factor de reflectancia (varia de 0 a 1 y se suele utilizar el 0.6)

i : Angulo de incidencia del sol [°]

c_{ps} : Ubicación del centro de presión solar [mm]

cg : Centro de gravedad

Campo magnético: El campo magnético ejerce un torque en el satélite, el cual se modela como se muestra a continuación, sabiendo que el campo magnético de la Tierra se puede aproximar tanto para una órbita polar como para una ecuatorial.

$$T_m = DB$$

$$B = \frac{2M}{R^3}\ (Polar); B = \frac{M}{R^3}\ (ecuatorial)$$

Donde:

D : Dipolo residual del satélite $[A \cdot m^2]$
B : Campo magnético de la Tierra [T]
M : Momento magnético de la Tierra ($7.96x10^{15}[T \cdot m^3]$)

R : Es la distancia desde el centro del dipolo (la Tierra) al satélite $[m]$

Anexo C: Sensores y actuadores utilizados

A modo de resumen se entregan tablas al inicio de cada sección de los componentes a modo de resumen según sus costos en masa, potencia y tamaño, en conjunto con su índice de performance característico por cada componente. Posterior a las tablas se muestra las especificaciones técnicas completas de los sensores y actuadores utilizados en conjunto con la referencia de su datasheet.

Giroscopio

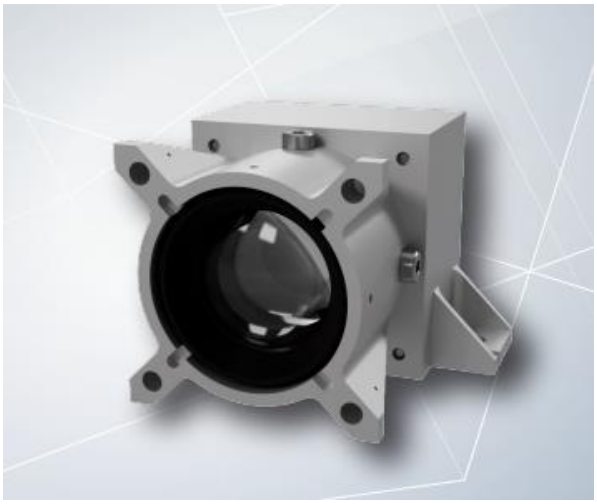
Tabla 17. Resumen de los giroscopios utilizados.

Nivel	Modelo	Masa [kg]	Potencia [W]	Tamaño [mm]	Estabilidad Bias [°/hr]	Ruido/σ [°/s]
Bajo	CRH03 – 200 (Silicon Sensing Systems)	0.045	0.15	47 x 33.5 x 25.4	0.05	0.12
Medio	CRH03 – 010 (Silicon Sensing Systems)	0.045	0.15	47 x 33.5 x 25.4	0.03	0.050
Alto	NSGY-001 (NewSpace System)	0.055	0.2	37.0 x 35.5 x 49.0	-	0.033

CRH03 – 200 y CRH03 – 010 (Silicon Sensing Systems) [45]

Bias Instability at 25°C	CRH03-010	–	0.03°/hr	–
	CRH03-025	–	0.04°/hr	–
	CRH03-100	–	0.04°/hr	–
	CRH03-200	–	0.05°/hr	–
	CRH03-400	–	0.10°/hr	–
Quiescent Noise	CRH03-010	–	0.050°/s rms	–
	CRH03-025	–	0.050°/s rms	–
	CRH03-100	–	0.12°/s rms	–
	CRH03-200	–	0.12°/s rms	–
	CRH03-400	–	0.12°/s rms	–

NSGY-001 (NewSpace System) [46]



	NSGY-001
FUNCTIONAL CHARACTERISTICS	
Rate estimation accuracy [3σ]	≤0.20 degrees/s (boresight) ≤0.05 degrees/s (cross-boresight)
Maximum slew rate	≥1.00 degrees/s
Detection capability	Mv ≥5.0
Maximum number of features tracked	15
Standard update rate	>1 Hz
Sky coverage	>99%
PHYSICAL CHARACTERISTICS	
Dimensions	37.0 mm x 35.5 mm x 49.0 mm
Mass	<55 g
ENVIRONMENTAL CHARACTERISTICS	
Thermal (operational)	-25 °C to +50 °C
Vibration (qualification)	14 g_{rms} (random)
INTERFACES	
Power supply	5 V_{DC}
Power Consumption	<200 mW (average)
Communication	SPI
Connector	nano-D (P15)
Mechanical	Front: 3 x M3 (w/ alignment slots) Top: 2 x M3 (w/ alignment slots)

Magnetómetros

Tabla 18. Resumen de los magnetómetros utilizados

Nivel	Modelo	Masa [kg]	Potencia [W]	Tamaño [mm]	Ruido máximo $\left[\frac{nT}{Hz}\right]$
Bajo	MM200 (AAC Clyde Space)	0.012	0.010	33 x 20 x 11.3	1.18
Medio	-	-	-	-	-
Alto	MAG-3 (AAC Clyde Space)	0.1	0.15-0.30	35.1x32.3x82.6	0.1-0.012

MM200 (AAC Clyde Space) [47]

TECHNICAL SPECIFICATIONS

Performance		
Sampling rate	Max 500	Hz (configurable)
Radiation tolerance	Up to 30	krad
Range	+800	μT
Total noise spectral density	1.18	nT/√Hz

Dimensions & Mass		
Flanged	33 x 20 x 11.3	mm
Minimal	20 x 20 x 11.3	mm
Mass(Flanged/Minimal)	12/10 ±1	g

Electrical specifications				
	Min.	Typ.	Max.	
Power Consumption	0.5	-	10	mA

MAG-3 (AAC Clyde Space) [48]

Performance Specifications	
Accuracy:	± 0.75% of Full Scale (0.5% typical)
Linearity:	± 0.015% of Full Scale (15 to 34 VDC input) ± 0.15% of Full Scale (5 V option)
Sensitivity:	100 μV/nT (other sensitivities available)
Scale Factor Temperature Shift:	0.007% Full Scale/°C Typical
Analog Output Options:	±10 Volts = ±100μT or ±5 Volts = ±60μT (other options available)
Axial Alignment:	Orthogonality Better Than ± 1 degree
Noise:	12 picoTesla RMS/ √Hz @1 Hz <100 picoTesla RMS/√ Hz @1 Hz (0 to 5 Volt Model)
Analog Output @ Zero Field:	± 0.025 Volt
Zero Shift with Temperature:	± 0.6 nT/°C
Susceptibility to Perming:	± 8 nT Shift with ± 5 Gauss Applied
Output Impedance:	332 Ω ± 5%
Frequency Response:	3 dB @ > 500 Hz (to > 4 kHz Wideband)
Over Load Recovery:	± 5 Gauss Slew < 2 ms

Electrical Specifications	
Input Voltage:	15 to 34 VDC or 5 Volt Regulated
Power Consumption:	Voltage Dependent (30 mA at any input voltage)
Connectors:	9 Pin Male "D" Type

Mechanical and Environmental	
Mass:	100 grams
Size:	3.51 cm x 3.23 cm x 8.26 cm
Operating Temperature:	-55°C to +85°C
Radiation:	> 10 Krad TID

Sun Sensor

Tabla 19. Resumen de los Sun Sensor utilizados

Nivel	Modelo	Masa [kg]	Potencia [W]	Tamaño [mm]	Exactitud/desv. estándar [°]
Bajo	CSS-01, CSS-02 (Space Micro)	0.010	0	12.7 diámetro x 9 altura	5/0.833
Medio	MSS-01(Space Micro)	0.036	0	34.9 diámetro x 24.3 altura	1/0.167
Alto	FSS (Bradford Space)	0.375	0.25	108 x 108 x 52.5	0.3/0.05

CSS-01, CSS-02 (Space Micro) [49]

SPECIFICATIONS

Field of View	120° full-angle circular field of view
Accuracy	±5° of 1-axis knowledge
Temperature Range	-40 to +93°C
Vibration Test Levels	14.1 grms
Shock Test Levels	60g protoqual
Interface	0 to 3.5 mA (typical) current sources on two flying leads: 50" (1.27m) in length, M22759/ 33-26, 26 AWG wire
Mounting	Three #2 through holes, 120° apart on a .700" (1.78 cm) diameter pattern (See Figure below).
Power	None required
Size	
Housing diameter	.500" (1.27cm)
Flange diameter	.900" (2.286 cm)
Sensor height	0.354" (.899 cm)
Volume	0.500" (1.27 cm) diameter × .354" (0.90 cm) height
Mass	0.022 lbs (10g) with 1.27 cm flying leads

MSS-01(Space Micro) [50]

	MSS-01	MSS-02
Field of View	48° full-angle circular field of view	120° full-angle circular field of view
Accuracy	±1° of 2-axis knowledge	
		(accuracy guaranteed within center 60° full-angle circular field of view)
Temperature Range	-40C to +93C	
Vibration Range	14.1g grms	
Shock Range	60g Protoqual	
Interface	Four 0 to 4.5mA (typical) current sources on five flying leads: 50 inch (1.27 m) in length, M22759/33-26, 26 AWG wire	
Mounting	Three #4 through holes, 120° apart on a 1.062 inch (2.70 cm) diameter pattern with two 0.064 diameter alignment holes (see figure below).	
Power	None required	
Housing diameter	.75" (1.91cm)	
Flange diameter	1.375" (3.49 cm)	
Sensor height	0.957" (2.43 cm)	0.407" (1.034 cm)
Volume	0.957" (2.43 cm) Height X 1.375 (3.49CM) Diameter	0.407" (1.034 cm) Height X 1.375 (3.49 cm) Diameter
Mass	0.08lbs (36 grams) with 1.27m flying leads	0.06lbs (25.5 grams) with 1.27m flying leads

FSS (Bradford Space) [51]

Characteristic	Performance / Interfaces Budget
Mass per sensor unit	Approximately 375 grams
Envelope dimension	<108 x 108 x 52.5 mm, including all protrusions by mounting feet, connectors, alignment cube and bondpin, exclusive dowel pins. Floor space: 104 x 94 mm
Nominal FOV	128° x 128° of the COTS sensor version
Unobstructed FOV (to be free of straylight sources)	138° x 138°
Accuracy	Two-axis measurement of solar aspect angles: EOL Bias error less than 0.3° (3 σ) in whole 128° x 128° FOV (throughout mission lifetime), after on-board implementation of ground calibration parameters. Note: This figure applies for condition without albedo
Resolution	Better than 0.03 degrees of arc
Noise equivalent angle	Less than 0.05° (3 σ)
Cross coupling between two axis read-outs	Change of $\Delta\alpha$ in one axis direction (α) may cause error $\Delta\beta < 0.05°$ in other direction (β).
Outputs	Analogue voltages in the range of 0-5V per quadrant; multiplexed on the basis of address command (rate should be about 1 ms).
Power consumption	< 0.25 W from ±15 V secondary power. No DC/DC converter included. (current per unit drawn from V+ approx 12 mA; current drawn from V- approx 3 mA)
Reliability	Failure rate: approx 70 FIT @30°C per unit; Reliability for 5 years in orbit for a single unit > 0.997.
Alignment	Alignment cube will be included; accuracy of faces of the cube: <10 arcsec; accuracy of alignment < 0.05 degrees of arc
Qualification temperature	-50 to +85 °C
Radiation Environment	The detector active element is made of p-type epitaxial silicon. With the shielding (>3 mm of cover glass thickness) the accumulated dose will be about 10 kRad. The degradation of the device output will be limited to less than 5% in current and an increase in temperature coefficient to 0.10%/°C max. Degradation will be common mode, common mode effects are cancelled out in sensor algorithms. EEE parts rad tolerance larger than 100 kRad, shielding > 7.5 mm aluminium. SEU and SEL Immune.
Electrical Stimuli / closed loop testing	Four voltage input interface provided, which by-passes the detector and enables closed loop testing.

Rueda de reacción

Tabla 20. Resumen de las ruedas de reacción utilizadas.

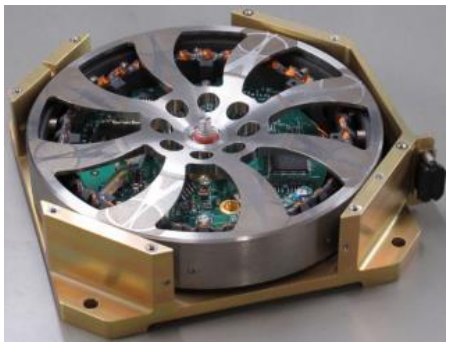
Nivel	Modelo	Masa [kg]	Potencia [W]	Tamaño [mm]	Peak Torque [Nm]; Capacidad de momento [Nms]
Bajo	CubeWheel small (CubesSpace)	0.060	0.65	28 x 28 x 26.2	0.00023; 0.00177
Medio	RW4-1.0 (RocketLab)	1.38	43	154 x 146 x 45	0.1; 1
Alto	RW4 (Blue Canyon Technologies)	3.2	10	108 x 108 x 52.5	0.250; 4

CubeWheel small (CubesSpace) [52]

CubeWheel S	
PERFORMANCE	
Speed Range [RPM]	±8000
Max Momentum [mNms]	1.77
Max Torque [mNm]	0.23
Static Imbalance [g-cm]	<0.003
Dynamic Imbalance [g-cm²]	< 0.005
PHYSICAL	
Mass [g]	60
Dimensions [mm]	28x28x26.2
Operating Temp. [°C]	

POWER & DATA	
Supply Voltage	
Average Power [mW]	150
Peak Power [W]	0.65
Communications	
QUALIFICATION	
Vibrations [g RMS]	14
Radiation	
ORDER INFO	
Unit Price	USD 5,170

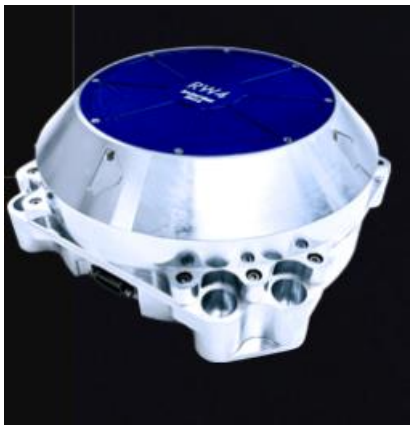
RW4-1.0 (RocketLab) [53]



SPECIFICATIONS

Momentum	Nominal: **1.0 Nms**
Torque	**±100 mNm at 0.8 Nms (at 28 V supply)**
Control Mode	**Speed or torque, with built-in control CPU**
Command / Telemetry	**Redundant RS-485, with galvanic isolation from primary**
Mechanical	Dimensions: **154 mm x 146 mm x 45 mm** Mass: **1380 g**
Supply Voltage	Nominal: **24 V to 34 V**, Maximum: **50 V**
Supply Power (28 V) (in vacuum)	**43 W @ 1.0 Nms, +60 mNm torque** **5.3 W @ 1.0 Nms, steady state** **1.4 W @ 0.2 Nms, steady state** **-28 W @ 1.0 Nms, −100 mNm regenerative braking**
Environment	Thermal: **−40°C to +70°C (operating)** Vibration: **>14 gRMS**
Radiation	**> 60 krad Total Dose. > 5 mm Al equivalent shield** **Heavy ion tested to 50 MeV-cm2/mg, LDRS tested** **Hardware TMR on all flip-flops, EDAC on all RAM**
Reliability	**Diamond coated hybrid ball bearings** **Redundant motor windings** **Active neutralization of rotor static charge**
Heritage	**Mechanical design from RW3-1.0, with 52 Units on-orbit.** **First flight deliveries December 2019**
Price	**US$80,000 each**

RW4 (Blue Canyon Technologies) [54]



General Parameters	
Torque	250 mNm
Angular Momentum	4 Nms
Mass	3.2 Kg
Supply Voltage	22 to 34 V
Power Consumption	10 W
Space Heritage	Yes
Interface	RS-422
Dimension	170 x 170 x 70 mm
Application	Spacecraft
Note	https://storage.googleapis.com/blue-canyon-tech-news/1/2022/04/BCT_DataSheet_Components_ReactionWheels.pdf

Magnetorquer

Tabla 21. Resumen de los magnetorquers utilizados.

Nivel	Modelo	Masa [kg]	Potencia [W]	Tamaño [mm]	Peak dipole [Am^2]
Bajo	CubeTorquer Coil small (CubeSpace)	0.028	0.42	18 x 14 x 62	0.24
Medio	NCTR-M012 (NewSpace systems)	0.053	0.8	94 x 15 x 13	1.19
Alto	MT15-1 (ZARM Technik)	0.4-0.55	1-1.55	329.5 largo x 17 diámetro	15

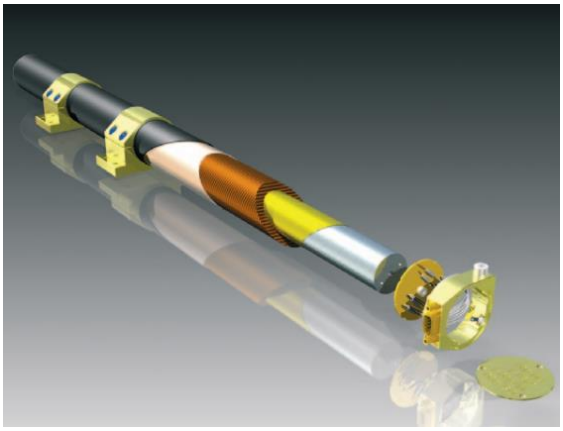
CubeTorquer Coil (CubeSpace) [52]

	CubeTorquer S
PERFORMANCE	
Magnetic Moment [Am²]	±0.24
Magnetic Gain [Am² /A]	2.8
Linearity (0-5V)	97.5%
PHYSICAL	
Mass [g]	28
Dimensions [mm]	18x14x62
POWER & DATA	
Resistance [Ω]	29-31
Max Current	
QUALIFICATION	
Vibrations	
Radiation	
ORDER INFO	
Unit Price	USD 870

NCTR-M012 (NewSpace systems) [55]

PERFORMANCE	NCTR-M003	NCTR-M012
FUNCTIONAL CHARACTERISTICS		
Magnetic moment (nominal)	0.29 Am^2	1.19 Am^2
Linearity (across operating range)	<± 5%	<± 5%
Residual moment	<0.01 Am^2	<0.01 Am^2
PHYSICAL CHARACTERISTICS		
Dimensions (l x w x h)	72 mm x 15 mm x 13 mm	94 mm x 15 mm x 13 mm
Mounting feet	2	2
Mass	<30 g	<53 g
Power (nominal)	<250 mW from 5 V supply	<800 mW nominal @ 5 V
ENVIRONMENTAL CHARACTERISTICS		
Thermal (acceptance)	-20 °C to +60 °C	-20 °C to +60 °C
Mechanical Tests (qualification)	21.06g_{RMS} (random)	21.06g_{RMS} (random)
Radiation (TID) (qualification)	n.a.	n.a.
INTERFACES		
Power supply	5 V_{DC}	5 V_{DC}
Data	n.a.	n.a.
Connector	Molex Pico-Lock	Molex Pico-Lock
Mechanical	4 x M2 Socket Head Cap Screws	4 x M2 Socket Head Cap Screws

MT15-1 (ZARM Technik) [56]



Off-the-Shelf Models						
Type	Linear Dipole Moment (Am^2)	Linear Voltage (V)	Linear Power (W)	Mass (kg)	Length (mm)	Dia. (mm)
MT2-1	2	5.0	0.5	0.2	157.5	15
MT5-2	5	5.0	0.77	0.3	240.0	18
MT6-2	6	5.0	0.5	0.3	325.0	14.5
MT10-2-H	10	10.0	1.0	0.35	330.0	17
MT10-2-AIR	10	11.0	1.1	2.7	1120 x 584 x 95 mm	
MT15-1	15	14.0	1.11	0.43	329.5	17

Anexo D: Cambios en los sistemas de referencia

De cuaternión a ángulos de Euler:

$$Roll = \arctan\left(\frac{2(q_3 q_0 + q_1 q_2)}{1 - 3(q_0^2 + q_1^2)}\right.$$

$$Pitch = \begin{cases} \arcsin(1.0), & si\ 2(q_3 q_1 - q_2 q_0) > 1.0 \\ \arcsin(-1.0), & si\ 2(q_3 q_1 - q_2 q_0) < -1.0 \\ \arcsin\big(2(q_3 q_1 - q_2 q_0)\big), & en\ otro\ caso \end{cases}$$

$$Yaw = \arctan\left(\frac{2(q_3 q_2 + q_0 q_1)}{1 - 2(q_1^2 + q_2^2)}\right)$$

De cuaterniones a DCM y viceversa:

$$DCM = \begin{bmatrix} q_3^2 + q_0^2 - q_2^2 - q_3^2 & 2(q_0 q_1 + q_2 q_3) & 2(q_0 q_2 - q_1 q_3) \\ 2(q_0 q_1 - q_2 q_3) & q_3^2 - q_0^2 + q_1^2 - q_2^2 & 2(q_1 q_2 + q_0 q_3) \\ 2(q_0 q_2 + q_1 q_3) & 2(q_1 q_2 - q_0 q_3) & q_3^2 - q_0^2 - q_1^2 + q_2^2 \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix}$$

$$q_0^2 = \frac{1}{4}(1 + C_{11} - C_{22} - C_{33})$$

$$q_1^2 = \frac{1}{4}(1 - C_{11} + C_{22} - C_{33})$$

$$q_2^2 = \frac{1}{4}(1 - C_{11} - C_{22} + C_{33})$$

$$q_3^2 = \frac{1}{4}(1 + C_{11} + C_{22} + C_{33})$$

Anexo E: Procedimiento a seguir para el uso del filtro de Kalman

En la sección 5.4.1 se introduce el filtro de Kalman extendido para estimar la orientación del satélite utilizando como condición inicial un cuaternión obtenido por el algoritmo TRIAD. Para iniciar el EKF se debe saber que el vector estado consiste en la parte vectorial del cuaternión y los componentes de la velocidad angular, sabiendo que la parte escalar del cuaternión es dependiente de las demás. Esto se hace con el objetivo de que el sistema sea observable, ya que generalmente los cuaterniones al presentar 4 componentes, se presentan 7 estados en el modelo espacio estado (EE), siendo que el rango de la matriz de observabilidad será de 6 incluyendo las velocidades angulares, ya que los modelos orbitales están solo en tres dimensiones.

$$q_3 = \sqrt{1 - (q_0)^2 - (q_1)^2 - (q_2)^2}$$

$$x = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ \omega_0 \\ \omega_1 \\ \omega_2 \end{bmatrix}$$

Como se mostró en el marco teórico, la dinámica de actitud del satélite es no lineal, la cual es la razón del uso del EKF. Las ecuaciones no lineales que representan la rotación del satélite se representan según el sistema espacio estado:

$$\dot{x} = f(x, u, t) + w$$

$$y = h(x, t) + v$$

Donde u es la entrada de control, w es el ruido del modelo, y es la salida, v es el ruido de salida, f es el modelo dinámico no lineal y el h es el modelo de medición. El f y h correspondientes a los modelos utilizados para el satélite se muestran a continuación.

$$f(x, u, t) = \begin{bmatrix} \dot{q} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \frac{1}{2}\left(\omega_2 q_1 - \omega_1 q_2 + \omega_0\sqrt{1 - (q_0)^2 - (q_1)^2 - (q_2)^2}\right) \\ \frac{1}{2}\left(-\omega_2 q_0 + \omega_0 q_2 + \omega_1\sqrt{1 - (q_0)^2 - (q_1)^2 - (q_2)^2}\right) \\ \frac{1}{2}\left(\omega_1 q_0 - \omega_0 q_1 + \omega_2\sqrt{1 - (q_0)^2 - (q_1)^2 - (q_2)^2}\right) \\ \frac{\omega_1\omega_2(I_y - I_z)}{I_x} + \frac{\tau_x}{I_x} \\ \frac{\omega_0\omega_2(I_x - I_z)}{I_y} + \frac{\tau_y}{I_y} \\ \frac{\omega_0\omega_1(I_x - I_y)}{I_z} + \frac{\tau_z}{I_z} \end{bmatrix}$$

$$h(x, t) = \begin{bmatrix} \hat{\beta} \\ \hat{\omega} \end{bmatrix}$$

Donde I_x, I_y y I_z son los momentos de inercia principales del satélite, τ_x, τ_y y τ_z son los torques externos, que para lazo abierto se consideran solo los de perturbación en altura LEO, y finalmente están $\hat{\beta}$ y $\hat{\omega}$ que son las mediciones de fuerza magnética usando el magnetómetro [15] y la velocidad angular medida por el giroscopio. Se asume solo como modelo de orientación el magnetómetro al ser más exacto que el sensor de sol, para así estimar de mejor manera la actitud del satélite.

Para obtener el estado a priori del sistema, se debe resolver la ecuación del modelo que se muestra a continuación. Se tiene en cuenta que para su resolución se utiliza el propagador numérico RK4 con un paso de tiempo adecuado para la convergencia.

$$x_k^- = \int f(x, u, t)\, dt$$

El siguiente paso es obtener las matrices jacobianas del modelo dinámico y de medición. Para ello se derivaron las ecuaciones respecto al vector estado, obteniendo F y H que representan cada una respectivamente.

$$F = \begin{bmatrix} -0.5\,\mathrm{q}_0\frac{\omega_0}{\mathrm{q}_3} & 0.5\omega_2 - 0.5\,\mathrm{q}_1\frac{\omega_0}{\mathrm{q}_3} & -0.5\omega_1 - 0.5\,\mathrm{q}_2\frac{\omega_0}{\mathrm{q}_3} & 0.5q_3 & -0.5q_2 & 0.5q_1 \\ 0.5\omega_2 - 0.5\,\mathrm{q}_0\frac{\omega_1}{\mathrm{q}_3} & -0.5\,\mathrm{q}_1\frac{\omega_1}{\mathrm{q}_3} & 0.5\omega_0 - 0.5\,\mathrm{q}_2\frac{\omega_1}{\mathrm{q}_3} & 0.5q_2 & 0.5q_3 & -0.5q_0 \\ 0.5\omega_1 - 0.5\,\mathrm{q}_0\frac{\omega_2}{\mathrm{q}_3} & -0.5\omega_0 - 0.5\,\mathrm{q}_1\frac{\omega_2}{\mathrm{q}_3} & -0.5\,\mathrm{q}_2\frac{\omega_2}{\mathrm{q}_3} & -0.5q_1 & 0.5q_0 & 0.5q_3 \\ 0 & 0 & 0 & \omega_2\frac{(I_x - I_z)}{I_y} & \omega_2\frac{(I_y - I_z)}{I_x} & \omega_1\frac{(I_y - I_z)}{I_x} \\ 0 & 0 & 0 & 0 & 0 & \omega_0\frac{(I_x - I_z)}{I_y} \\ 0 & 0 & 0 & \omega_1\frac{(I_x - I_y)}{I_z} & \omega_0\frac{(I_x - I_y)}{I_z} & 0 \end{bmatrix}$$

$$H = \begin{bmatrix} 2I_{3x3} - \dfrac{2\hat{\beta}\beta^T}{\|\hat{\beta}\|\|\beta\|} & O_{3x3} \\ O_{3x3} & I_{3x3} \end{bmatrix}$$

De la matriz H se sabe que la primera componente es obtenida de Vélez et al. [15], basada en la utilización de un solo vector de observación y de referencia utilizando las fuerzas magnéticas en el algoritmo TRIAD. Además, O_{3x3} es una matriz cuadrada de 3x3 de ceros, mientras que I_{3x3} es la matriz identidad de 3x3.

Por otro lado, el ruido del modelo y de la salida se representan mediante las matrices Q y R mostradas a continuación:

$$Q = [I_{6X6}]$$

$$R = \begin{bmatrix} \sigma^2_{\text{magn}} & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma^2_{\text{magn}} & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma^2_{\text{magn}} & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma^2_{\text{gyros}} & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma^2_{\text{gyros}} & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma^2_{\text{gyros}} \end{bmatrix}$$

Siendo σ la desviación estándar o el error correspondiente al sensor, que en este caso puede ser el magnetómetro o el giroscopio.

Con esto, se puede obtener la matriz de covarianza de error a priori mediante la siguiente formula, sabiendo que la primera matriz de covarianza de error posteriori es una condición inicial dada como una matriz identidad de 6x6.

$$P_k^- = F P_k^+ F^T + Q_k$$

Luego, se obtiene la ganancia de Kalman mediante la siguiente relación:

$$K_k = P_k^- H_k^T (R_k + H_k P_k^- H_k^T)^{-1}$$

Además, se tiene ν que es el residual de la medición o la "innovación", que para el caso del magnetómetro también fue obtenida de Vélez et al. [15], mientras que para el giroscopio se asume la resta entre el modelo de velocidad angular y la medida del giroscopio.

$$\nu_k = \hat{y}_k - h(x_k^-, t) = \begin{bmatrix} \dfrac{\hat{\beta} \times \beta}{\|\hat{\beta}\| \|\beta\|} \\ \widehat{\omega} - \omega \end{bmatrix}$$

Posteriormente, se tiene que para la obtención del vector estado a posteriori, se debe conocer la perturbación Δx_k, la cual se conoce mediante la siguiente relación:

$$\Delta x_k = K_k \nu_k$$

Con esto, se tiene que para la velocidad angular a posteriori se considera la suma común para esta etapa. Por otro lado, se implementa la multiplicación de cuaterniones para tener el a posteriori en la orientación. Dichas relaciones se muestran a continuación:

$$\omega_k^+ = \Delta\omega_k + \omega_k^-$$

$$q_k^+ = \left[\frac{q_k^-}{\sqrt{1 - \left(q_{0,k}^-\right)^2 - \left(q_{1,k}^-\right)^2 - \left(q_{2,k}^-\right)^2}} \right] \otimes \left[\frac{\Delta q_k}{\sqrt{1 - \Delta q_{0,k}^2 - \Delta q_{1,k}^2 - \Delta q_{2,k}^2}} \right]$$

$$x_k^+ = \begin{bmatrix} q_k^+ \\ \omega_k^+ \end{bmatrix}$$

Finalmente, se actualiza la matriz de covarianza de error a posteriori utilizando la siguiente relación:

$$P_k^+ = (I_{6x6} - K_k H_k) P_k^-$$

Con esto, se puede propagar desde el paso k al paso k+1 hasta el tiempo final de simulación. Faltó implementar esto de manera ordenada y funcional dentro de la suite de simulación, la cual se realizará en trabajo futuro.

Anexo F: Procedimiento para el diseño del control lineal del *CubeSat*

Primero se plantean las ecuaciones de la cinemática del cuaternión utilizando la componente escalar (q_3) como dependiente de las componentes vectoriales, sabiendo que siempre la norma del cuaternión es igual a 1, dejando la siguiente expresión:

$$q_3 = \sqrt{1 - (q_0)^2 - (q_1)^2 - (q_2)^2}$$

Con ello, se obtiene el $f(x)$ mostrado en el Anexo E, sabiendo que en las tres primeras ecuaciones las tres componentes de la velocidad angular están entre los marcos de referencia de control (del cuerpo) y orbital (RPY o requerido), mientras que las tres últimas tienen la velocidad angular entre los marcos de referencia de control e inercial (ECI).

$$\dot{q}_0 = \frac{1}{2}\left(\omega_{2\_co} q_1 - \omega_{1\_co} q_2 + \omega_{0\_co}\sqrt{1 - (q_0)^2 - (q_1)^2 - (q_2)^2}\right) \tag{10}$$

$$\dot{q}_1 = \frac{1}{2}\left(-\omega_{2\_co} q_0 + \omega_{0\_co} q_2 + \omega_{1\_co}\sqrt{1 - (q_0)^2 - (q_1)^2 - (q_2)^2}\right) \tag{11}$$

$$\dot{q}_2 = \frac{1}{2}\left(\omega_{1\_co} q_0 - \omega_{0\_co} q_1 + \omega_{2\_co}\sqrt{1 - (q_0)^2 - (q_1)^2 - (q_2)^2}\right) \tag{12}$$

$$\dot{\omega}_{0,ci} I_x = \omega_{1,ci}\omega_{2,ci}\left(I_y - I_z\right) + \tau_{x,ctrl} + \tau_{x,pert} \tag{13}$$

$$\dot{\omega}_{1,ci} I_y = \omega_{0,ci}\omega_{2,ci}(I_x - I_z) + \tau_{y,ctrl} + \tau_{y,pert} \tag{14}$$

$$\dot{\omega}_{2,ci} I_z = \omega_{0\_ci}\omega_{1\_ci}\left(I_x - I_y\right) + \tau_{z,ctrl} + \tau_{z,pert} \tag{15}$$

Primero, se debe conocer que los torques de perturbación en el espacio τ_{pert} son los que están representados en el peor caso para cada uno en el Anexo B, mientras que el torque de control ejercido por el magnetorquer se expresa de la siguiente manera [14]:

$$\tau_{ctrl} = \frac{\bar{m} \times \bar{b}}{\|b\|} \times \bar{b}$$

Sabiendo que $\bar{b}$ son las fuerzas magnéticas en el marco de referencia del cuerpo y $\bar{m}$ es el momento dipolar del magnetorquer. Esta ecuación esta descompuesta en sus tres componentes como se muestra a continuación:

$$\tau_{x,ctrl} = \left(\frac{b_x m_z - b_z m_x}{\|b\|}\right) b_z - \left(\frac{b_y m_x - b_x m_y}{\|b\|}\right) b_y$$

$$\tau_{y,ctrl} = -\left(\frac{b_z m_y - b_y m_z}{\|b\|}\right) b_z + \left(\frac{b_y m_x - b_x m_y}{\|b\|}\right) b_x$$

$$\tau_{z,ctrl} = \left(\frac{b_z m_y - b_y m_z}{\|b\|}\right) b_y - \left(\frac{b_x m_z - b_z m_x}{\|b\|}\right) b_x$$

Por último, antes de linealizar el sistema según las variables de estado $x = [q_0, q_1, q_2, \omega_{0,co}, \omega_{1,co}, \omega_{2,co}]$ y las variables de control $u = [m_x, m_y, m_z]$, todas las ecuaciones deben estar en el mismo sistema de referencia, por lo que se debe aplicar un reemplazo en las ecuaciones (13), (14) y (15) para que estén en el sistema de referencia de control vs orbita. Para ello se implementa el siguiente cambio de variable propuesto por Torczynski [14], sabiendo que $\omega_{0,o}$ es el "orbital period" correspondiente a la velocidad angular en el marco de referencia orbital:

$$\begin{bmatrix}\omega_{0,ci}\\ \omega_{1,ci}\\ \omega_{2,ci}\end{bmatrix} = \begin{bmatrix}\omega_{0,co}\\ \omega_{1,co}\\ \omega_{2,co}\end{bmatrix} + \begin{bmatrix} q_3^2+q_0^2-q_1^2-q_2^2 & 2(q_0q_1+q_2q_3) & 2(q_0q_2-q_1q_3)\\ 2(q_0q_1-q_2q_3) & q_3^2-q_0^2+q_1^2-q_2^2 & 2(q_1q_2+q_0q_3)\\ 2(q_0q_2+q_1q_3) & 2(q_1q_2-q_0q_3) & q_3^2-q_0^2-q_1^2+q_2^2\end{bmatrix} \cdot \begin{bmatrix}\omega_{0,o}\\ 0\\ 0\end{bmatrix}$$

Al implementar dicho cambio de variable, las ecuaciones de la conservación de momentum angular quedan representados de la siguiente manera:

$$\dot{\omega}_{0,co} = \left(\omega_{1,co} + \omega_{0,o}\left[2\left(q_0q_1 - q_2\sqrt{1-(q_0)^2-(q_1)^2-(q_2)^2}\right)\right]\right) \cdot \left(\omega_{2,co} + \omega_{0,o}\left[2\left(q_0q_2 + q_1\sqrt{1-(q_0)^2-(q_1)^2-(q_2)^2}\right)\right]\right) \cdot \frac{I_y - I_z}{I_x} - \omega_{0,o}I_x[-4q_1\dot{q}_1 - 4q_2\dot{q}_2] + \frac{\tau_{x,ctrl}}{I_x} + \frac{\tau_{x,pert}}{I_x}$$

$$\dot{\omega}_{1,co} = \left(\omega_{0,co} + \omega_{0,o}[1-2q_1^2-2q_2^2]\right) \cdot \left(\omega_{2,co} + \omega_{0,o}\left[2\left(q_0q_2 + q_1\sqrt{1-(q_0)^2-(q_1)^2-(q_2)^2}\right)\right]\right) \cdot \frac{I_x - I_z}{I_y} - \omega_{0,o}I_y\left[2\left(\dot{q}_0q_1 + q_0\dot{q}_1 - \dot{q}_2\sqrt{1-(q_0)^2-(q_1)^2-(q_2)^2} + q_2\frac{q_0\dot{q}_0 + q_1\dot{q}_1 + q_2\dot{q}_2}{\sqrt{1-(q_0)^2-(q_1)^2-(q_2)^2}}\right)\right] + \frac{\tau_{y,ctrl}}{I_y} + \frac{\tau_{y,pert}}{I_y}$$

$$\dot{\omega}_{2,co} = \left(\omega_{0,co} + \omega_{0,o}[1-2q_1^2-2q_2^2]\right) \cdot \left(\omega_{1,co} + \omega_{0,o}\left[2\left(q_0q_1 - q_2\sqrt{1-(q_0)^2-(q_1)^2-(q_2)^2}\right)\right]\right) \cdot \frac{I_x - I_y}{I_z} - \omega_{0,o}I_z\left[2\left(\dot{q}_0q_2 + q_0\dot{q}_2 + \dot{q}_1\sqrt{1-(q_0)^2-(q_1)^2-(q_2)^2} - q_1\frac{q_0\dot{q}_0 + q_1\dot{q}_1 + q_2\dot{q}_2}{\sqrt{1-(q_0)^2-(q_1)^2-(q_2)^2}}\right)\right] + \frac{\tau_{z,ctrl}}{I_z} + \frac{\tau_{z,pert}}{I_z}$$

Con esta representación, se obtiene nuevamente el jacobiano F que será utilizado tanto para el lazo cerrado del EKF, como para el control lineal mostrado en esta sección. Sabiendo que las primera tres ecuaciones dinámicas no varían, se mostraran los cambios a F por componente en las ecuaciones necesarias. Se debe tener en cuenta que F_{41} y F_{52} provienen del torque de perturbación externa presente a través del gradiente de gravedad, el cual se relaciona como se muestran en las relaciones según Torczynski [14].

$$F_{41} = 6 * \omega_{0,o}^2 * (I_x - I_y)$$

$$F_{42} = F_{43} = F_{44} = 0$$

$$F_{45} = \left(\omega_2 + \omega_{0,o} * (2q_0q_2 - 2q_1q_3)\right) * \frac{I_y - I_z}{I_x} - \omega_{0,o}I_x * (-2q_1q_3 + 2q_2q_1)$$

$$F_{46} = \left(\omega_1 + \omega_{0,o} * (2q_0q_1 - 2q_2q_3)\right) * \frac{I_y - I_z}{I_x} - \omega_{0,o} I_x * (2q_1q_0 - 2q_2q_3)$$

$$F_{51} = F_{53} = 0$$

$$F_{52} = 6 * \omega_{0,o}^2 * (I_z - I_y)$$

$$F_{54} = \left(\omega_2 + \omega_{0,o}(2q_0q_2 + 2q_1q_3)\right) * \frac{I_x - I_z}{I_y} - \omega_{0,o} I_y \left(q_3q_1 + q_2q_0 - q_1q_3 + \frac{q_2q_0q_3 + q_2q_1q_2 + q_2q_2q_1}{q_3}\right)$$

$$F_{55} = -\omega_{0,o} I_y \left(-q_2q_1 + q_3q_0 - q_0q_3 + \frac{-q_2q_0q_2 + q_2q_1q_3 + q_2q_2q_0}{q_3}\right)$$

$$F_{56} = \left(\omega_0 + \omega_{0,o}(1 - 2q_1^2 - 2q_2^2)\right) * \frac{I_x - I_z}{I_y} - \omega_{0,o} I_y \left(q_1q_1 + q_0q_0 - q_3q_3 + \frac{q_2q_0q_1 - q_2q_1q_0 + q_2q_2q_3}{q_3}\right)$$

$$F_{61} = F_{62} = F_{63} = 0$$

$$F_{64} = \left(\omega_1 + \omega_{0,o}(2q_0q_1 - 2q_2q_3)\right) * \frac{I_x - I_y}{I_z} - \omega_{0,o} I_z \left(q_3q_2 - q_1q_0 + q_2q_3 - \frac{q_1q_0q_3 + q_1q_1q_2 - q_1q_2q_1}{q_3}\right)$$

$$F_{65} = \left(\omega_0 + \omega_{0,o}(1 - 2q_1^2 - 2q_2^2)\right) * \frac{I_x - I_y}{I_z} - \omega_{0,o} I_z \left(-q_2q_2 - q_0q_0 + q_3q_3 - \frac{q_1q_0q_2 + q_1q_1q_3 - q_1q_2q_0}{q_3}\right)$$

$$F_{66} = -\omega_{0,o} I_z \left(-q_1q_2 + q_3q_0 - q_0q_3 - \frac{q_1q_0q_1 - q_1q_1q_0 + q_1q_2q_3}{q_3}\right)$$

Anexo G: Implementación de los resultados de los MoP de apuntamiento

Para mostrar el paso a paso de la cuantificación de los MoP de apuntamiento, se utilizará como ejemplo la simulación del SUCHAI-3 con los datos utilizados en el Capítulo 6, con los componentes físicos de nivel alto.

Jitter

Para la cuantificación de este MoP de apuntamiento, en primera instancia se aplica el filtro pasa alto con una frecuencia tope de 10 Hz, mostrando solo las señales mayores a dicho valor. Con esto se observa la Figura 32, asentándose la señal siempre en el cero con el ruido de las vibraciones del sistema.

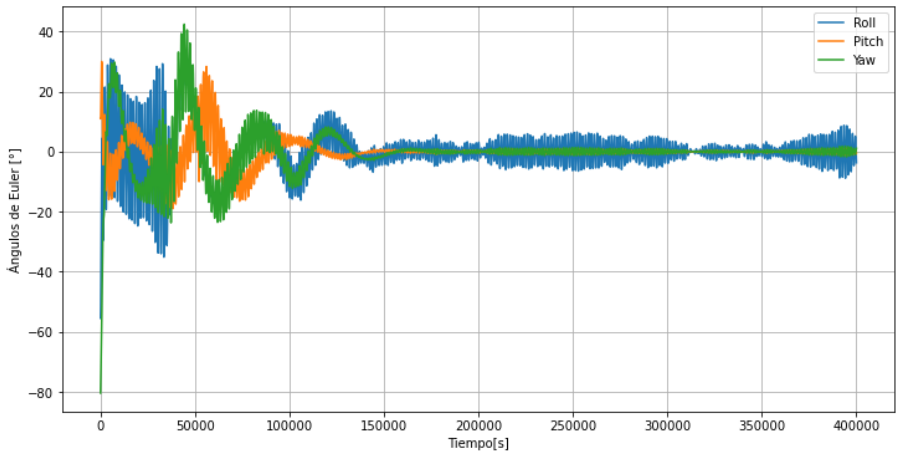


Figura 32. Filtro pasa alto de 10 Hz en la respuesta al sistema de sensores y actuadores de nivel alto.

Con esto, se obtienen las frecuencias y sus densidades espectro potencias aproximadas mediante la función welch de scipy.signal disponible en Python. Posteriormente, se selecciona un ancho de banda bajo, para obtener en dichos intervalos el PSD que representa el *jitter* del sistema. Esto se observa en los ángulos de Euler en las Figura 33, Figura 34 y Figura 35 para cada uno, presentado mayores valores a menores frecuencias. Se observa en el área pintada la integración dentro del ancho de banda realizada.

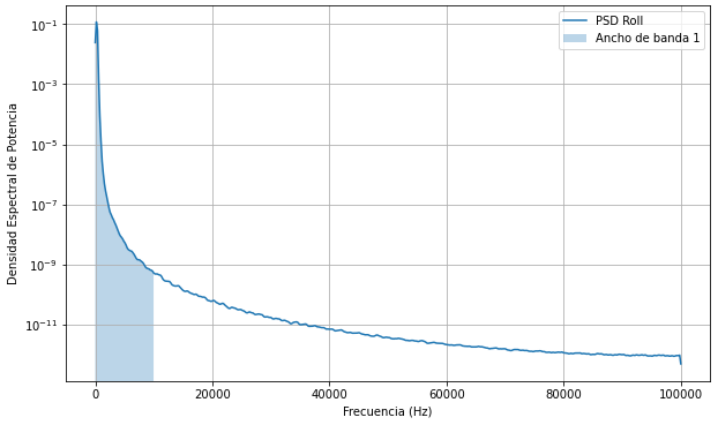


Figura 33. Densidad espectro potencia en ancho de banda seleccionada para Roll.

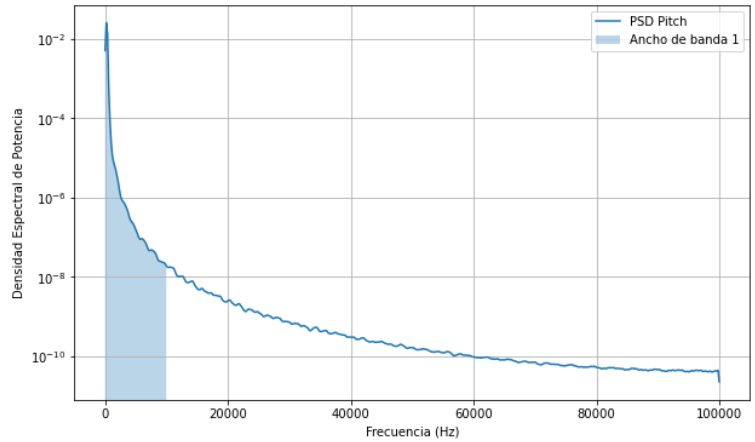


Figura 34. Densidad espectro potencia en ancho de banda seleccionada para Pitch.

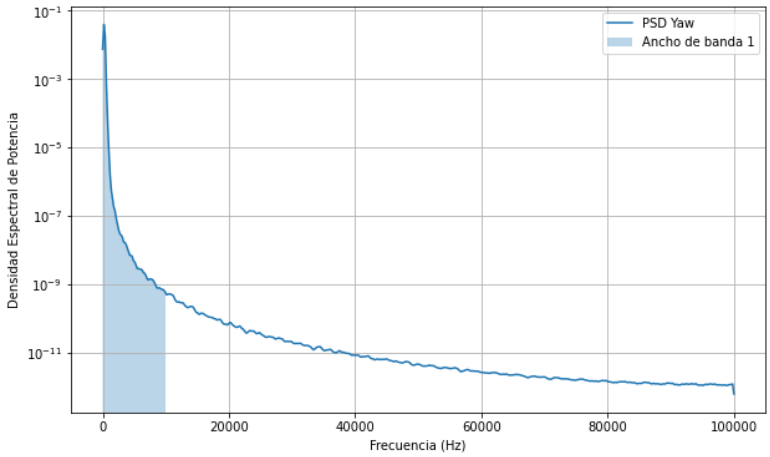


Figura 35. Densidad espectro potencia en ancho de banda seleccionada para Yaw.

Agilidad

Por otro lado, para la cuantificación de la agilidad, se aplica un filtro pasa bajo de 10 Hz, para así obtener la respuesta de baja frecuencia, que es la que representa al control de la actitud sin las vibraciones del *jitter*. Esto se hace con el objetivo de conocer lo más preciso posible el tiempo de asentamiento con una banda del 5%.

En las Figura 36, Figura 37 y Figura 38 se muestran los ángulos de Euler Roll, Pitch y Yaw respectivamente una vez aplicado el filtro pasa bajo, en conjunto con una gráfica acercada en la zona de asentamiento, mostrando además los limites superior e inferior dadas por la banda de asentamiento. Una vez que la orientación del satélite está presente dentro de la banda de asentamiento, el tiempo inicial para el ingreso sin retirarse es el tiempo de asentamiento con el cual se cuantifica la agilidad.

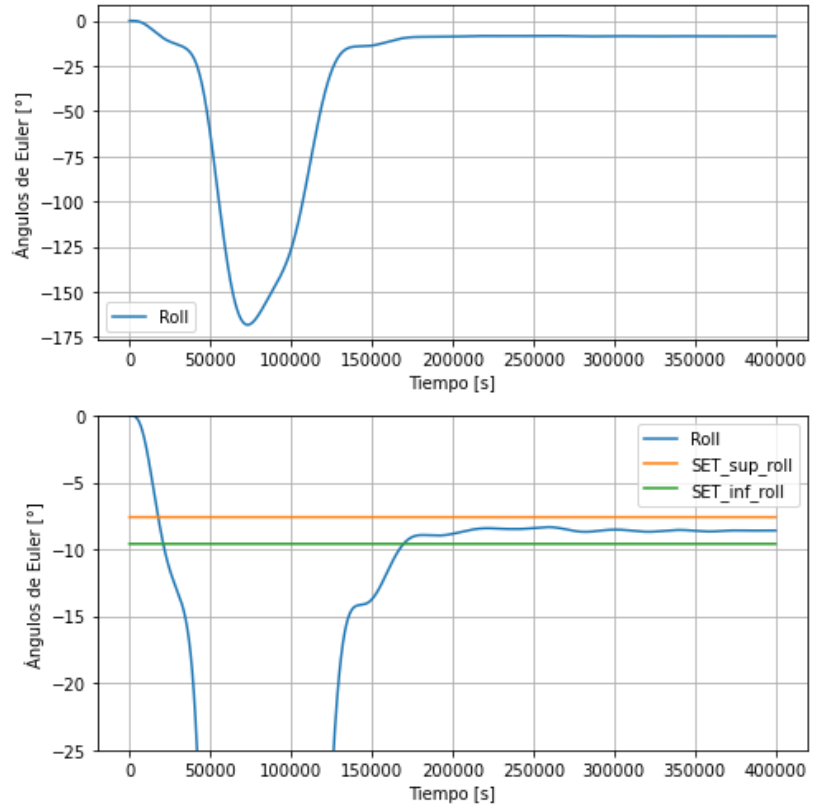


Figura 36. Banda de asentamiento Roll en filtro pasa bajo.

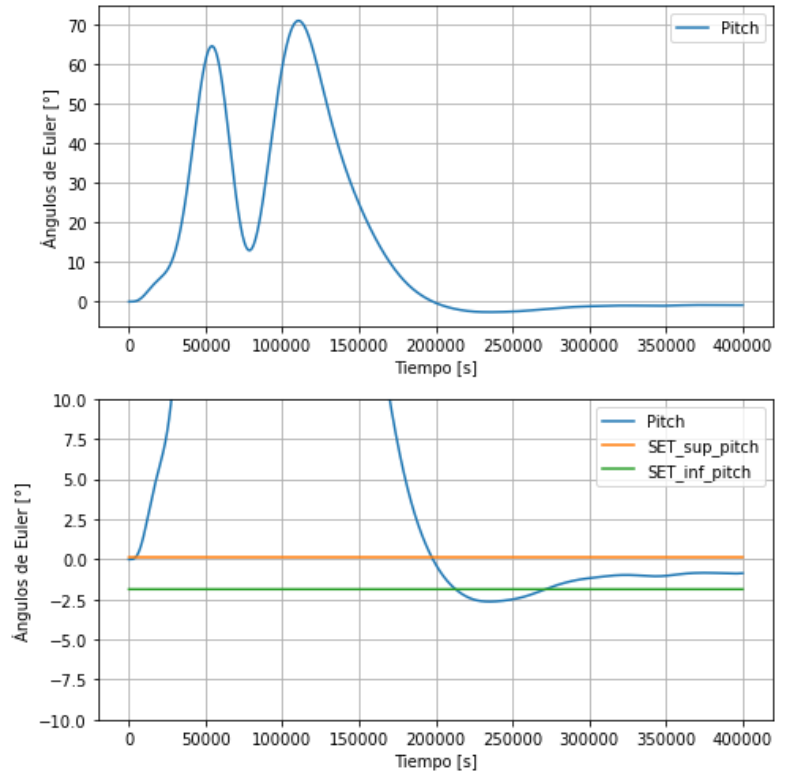


Figura 37. Banda de asentamiento Pitch en filtro pasa bajo.

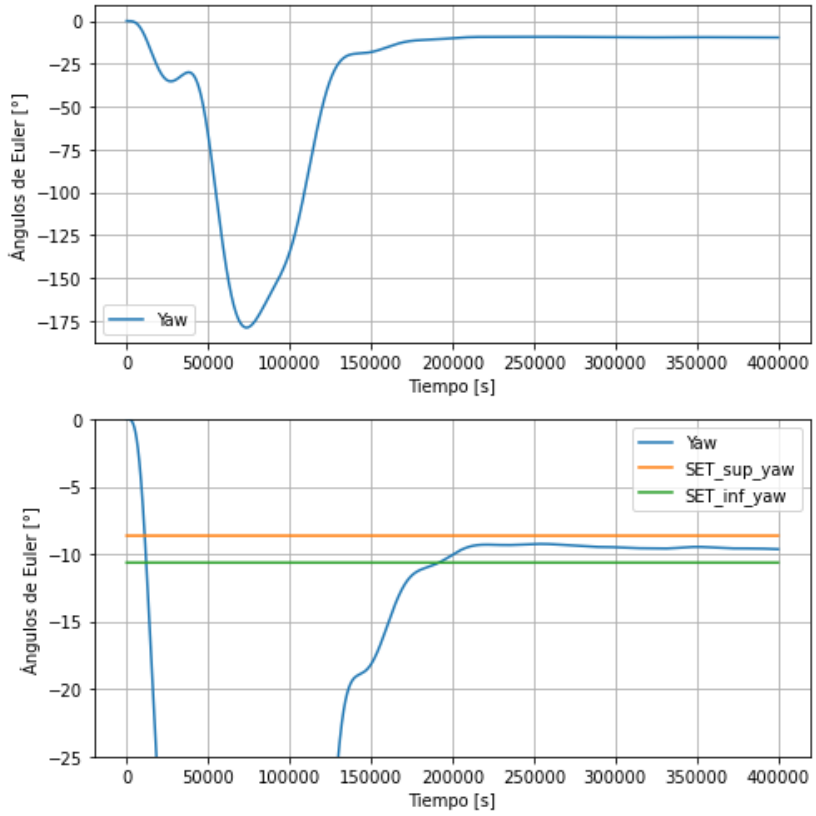


Figura 38. Banda de asentamiento Yaw en filtro pasa bajo.

Exactitud de apuntamiento

Finalmente, para la exactitud de apuntamiento en sus tres componentes se busca utilizar los valores de los ángulos de Euler obtenidos después del tiempo de asentamiento del filtro pasa bajo con tope de 10 Hz. Posteriormente se calcula el promedio de dichos valores para obtener el grado sexagesimal aproximado que representa con que error se está apuntando hacia el centro de la Tierra.

Anexo H: Código Python simulador

Se presenta el github con los códigos necesarios para la realización de la simulación, cuyo paso a paso de uso se presenta en el README.rd presente en el mismo URL.

https://github.com/mtacul/Thesis_project.git

Además, se presentan en su formato nativo a continuación. En primera instancia, se muestra Functions.py en el cual se presentan todas las funciones creadas para la suite de simulación, las cuales son llamadas para su uso en cual otra librería (se debe descargar la carpeta pyIGRF para el uso de los script de Python)

Functions.py.

```
from skyfield.positionlib import Geocentric
import numpy as np
from datetime import datetime
import math
from scipy.signal import butter, lfilter

def eci2lla(posicion, fecha):
    from skyfield.api import Distance, load, utc, wgs84
    ts = load.timescale()
    fecha = fecha.replace(tzinfo=utc)
    t = ts.utc(fecha)
    d = [Distance(m=i).au for i in (posicion[0]*1000, posicion[1]*1000, posicion[2]*1000)]
    p = Geocentric(d,t=t)
    g = wgs84.subpoint(p)
    latitud = g.latitude.degrees
    longitud = g.longitude.degrees
    altitud = g.elevation.m
    return latitud, longitud, altitud

#%%inversa de un cuaternion

def inv_q(q):
    inv_q = np.array([-q[0],-q[1],-q[2],q[3]])
    return inv_q

#%% Para vector sol

def datetime_to_jd2000(fecha):
    t1 = (fecha - datetime(2000, 1, 1, 12, 0, 0)).total_seconds()
    t2 = 86400  # Número de segundos en un día
```

```
    jd2000 = t1 / t2
    return jd2000

def sun_vector(jd2000):
    M_sun = 357.528 + 0.9856003*jd2000
    M_sun_rad = M_sun * np.pi/180
    lambda_sun = 280.461 + 0.9856474*jd2000 + 1.915*np.sin(M_sun_rad)+0.020*np.sin(2*M_sun_rad)
    lambda_sun_rad = lambda_sun * np.pi/180
    epsilon_sun = 23.4393 - 0.0000004*jd2000
    epsilon_sun_rad = epsilon_sun * np.pi/180
    X_sun = np.cos(lambda_sun_rad)
    Y_sun = np.cos(epsilon_sun_rad)*np.sin(lambda_sun_rad)
    Z_sun = np.sin(epsilon_sun_rad)*np.sin(lambda_sun_rad)
    return X_sun, Y_sun, Z_sun

#%% TRIAD solution

def TRIAD(V1,V2,W1,W2):
    r1 = V1
    r2 = np.cross(V1,V2) / np.linalg.norm(np.cross(V1,V2))
    r3 = np.cross(r1,r2)
    M_obs = np.array([r1,r2,r3])
    s1 = W1
    s2 = np.cross(W1,W2) / np.linalg.norm(np.cross(W1,W2))
    s3 = np.cross(s1,s2)
    M_ref = np.array([s1,s2,s3])

    A = np.dot(M_obs,np.transpose(M_ref))
    return A

#%% de cuaternion a angulos de euler
def quaternion_to_euler(q):
    # Extracción de los componentes del cuaternión
    x, y, z, w = q

    # Cálculo de ángulos de Euler en radianes
    t0 = +2.0 * (w * x + y * z)
    t1 = +1.0 - 2.0 * (x * x + y * y)
    roll_x = math.atan2(t0, t1)
```

```
    t2 = +2.0 * (w * y - z * x)
    t2 = +1.0 if t2 > +1.0 else t2
    t2 = -1.0 if t2 < -1.0 else t2
    pitch_y = math.asin(t2)

    t3 = +2.0 * (w * z + x * y)
    t4 = +1.0 - 2.0 * (y * y + z * z)
    yaw_z = math.atan2(t3, t4)

    # Convierte los ángulos a grados si lo deseas
    roll_deg = np.degrees(roll_x)
    pitch_deg = np.degrees(pitch_y)
    yaw_deg = np.degrees(yaw_z)

    return roll_deg, pitch_deg, yaw_deg


def quat_mult(qk_priori,dqk):

    dqk_n = dqk

# Realizar la multiplicación de cuaterniones
    result = np.array([
    qk_priori[3]*dqk_n[0]  +  qk_priori[0]*dqk_n[3]  +  qk_priori[1]*dqk_n[2]  -
qk_priori[2]*dqk_n[1],  # Componente i
    qk_priori[3]*dqk_n[1]  +  qk_priori[1]*dqk_n[3]  +  qk_priori[2]*dqk_n[0]  -
qk_priori[0]*dqk_n[2],  # Componente j
    qk_priori[3]*dqk_n[2]  +  qk_priori[2]*dqk_n[3]  +  qk_priori[0]*dqk_n[1]  -
qk_priori[1]*dqk_n[0],  # Componente k
    qk_priori[3]*dqk_n[3] - qk_priori[0]*dqk_n[0] - qk_priori[1]*dqk_n[1] - qk_priori[2]*dqk_n[2]
# Componente escalar
    ])
    return result

#%%Funciones para sensores

def simulate_magnetometer_reading(B_eci, ruido):
    # np.random.seed(42)  # Puedes cambiar el número 42 a cualquier otro número entero

    # Simular el ruido gaussiano
    noise = np.random.normal(0, ruido, 1)
```

```
    # Simular la medición del magnetómetro con ruido
    measurement = B_eci + noise


    return measurement

# Obtener la desviacion estandar del sun sensor
def sigma_sensor(acc):
    sigma = acc/(2*3)
    return sigma

# Funcion para generar realismo del sun sensor
def simulate_sunsensor_reading(vsun,sigma):
    # np.random.seed(42)  # Puedes cambiar el número 42 a cualquier otro número entero

    sigma_rad = sigma*np.pi/180

    # Simulación de la medición con error
    error = np.random.normal(0, sigma_rad, 1)  # Genera un error aleatorio dentro de la precisión del
sensor

    measured_vsun = vsun + error

    return measured_vsun

# Funcion para generar realismo del giroscopio
def simulate_gyros_reading(w,ruido,bias):
    # np.random.seed(42)  # Puedes cambiar el número 42 a cualquier otro número entero

    #aplicar el ruido del sensor
    noise = np.random.normal(0, ruido, 1)

    #Simular la medicion del giroscopio
    measurement = w + noise + bias

    return measurement

#%% matrices A y B control PD lineal

def A_PD(I_x,I_y,I_z,w0_O, w0,w1,w2):
```

```
    A1 = np.array([0, 0.5*w2, -0.5*w1, 0.5, 0,0])
    A2 = np.array([-0.5*w2,0,0.5*w0,0,0.5,0])
    A3 = np.array([0.5*w1,-0.5*w0,0,0,0,0.5])
    A4 = np.array([6*w0_O**2*(I_x-I_y), 0, 0, 0, w2*(I_y-I_z)/I_x, w1*(I_y-I_z)/I_x])
    A5 = np.array([0, 6*w0_O**2*(I_z-I_y), 0, w2*(I_x-I_z)/I_y,0, (w0+w0_O)*(I_x-I_z)/I_y + I_y*w0_O])
    A6 = np.array([0, 0, 0, w1*(I_y-I_x)/I_z, (w0+w0_O)*(I_y-I_x)/I_z - I_z*w0_O, 0])

    A_k = np.array([A1,A2,A3,A4,A5,A6])

    return A_k

def B_PD(I_x,I_y,I_z,B_magnet):
    b_norm = np.linalg.norm(B_magnet)
    B123 = np.zeros((3,3))
    B4 = np.array([(-(B_magnet[2]**2)-B_magnet[1]**2)/(b_norm*I_x), B_magnet[1]*B_magnet[0]/(b_norm*I_x), B_magnet[2]*B_magnet[0]/(b_norm*I_x)])
    B5 = np.array([B_magnet[0]*B_magnet[1]/(b_norm*I_y), (-B_magnet[2]**2-B_magnet[0]**2)/(b_norm*I_y), B_magnet[2]*B_magnet[1]/(b_norm*I_y)])
    B6 = np.array([B_magnet[0]*B_magnet[2]/(b_norm*I_z), B_magnet[1]*B_magnet[2]/(b_norm*I_z), (-B_magnet[1]**2-B_magnet[0]**2)/(b_norm*I_z)])

    B_k = np.vstack((B123,B4,B5,B6))
    #B_k = np.array([B123,B4,B5,B6])

    return B_k

def rk4_step_PD(dynamics, x, A, B, u, h):
    k1 = h * dynamics(A, x, B, u)
    k2 = h * dynamics(A, x + 0.5 * k1, B, u)
    k3 = h * dynamics(A, x + 0.5 * k2, B, u)
    k4 = h * dynamics(A, x + k3, B, u)

    # Update components of q
    q0_new = x[0] + (k1[0] + 2 * k2[0] + 2 * k3[0] + k4[0]) / 6
    q1_new = x[1] + (k1[1] + 2 * k2[1] + 2 * k3[1] + k4[1]) / 6
    q2_new = x[2] + (k1[2] + 2 * k2[2] + 2 * k3[2] + k4[2]) / 6

    q_new_real = np.array([q0_new, q1_new, q2_new])

    # Update components of w
```

```
    w0_new = x[3] + (k1[3] + 2 * k2[3] + 2 * k3[3] + k4[3]) / 6
    w1_new = x[4] + (k1[4] + 2 * k2[4] + 2 * k3[4] + k4[4]) / 6
    w2_new = x[5] + (k1[5] + 2 * k2[5] + 2 * k3[5] + k4[5]) / 6
    w_new = np.array([w0_new, w1_new, w2_new])

    return q_new_real, w_new

def dynamics(A, x, B, u):
    return np.dot(A, x) - np.dot(B, u)

def K(Kp_x, Kp_y, Kp_z, Kd_x, Kd_y, Kd_z):
    K_gain = np.hstack((np.diag([Kp_x,Kp_y,Kp_z]), np.diag([Kd_x,Kd_y,Kd_z])))
    return K_gain

def quaternion_to_dcm(q):
    x, y, z, w = q
    dcm = np.array([
        [w**2 + x*2 + 2*y**2 + 2*z**2, 2*x*y + 2*w*z, 2*x*z - 2*w*y],
        [2*x*y - 2*w*z, w**2-x**2+y**2+z**2, 2*y*z + 2*w*x],
        [2*x*z + 2*w*y, 2*y*z - 2*w*x, w**2-x**2-y**2+z**2]
    ])
    return dcm

def high_pass_filter(signal, cutoff_freq, sample_rate, order=4):
    nyquist = 0.5 * sample_rate
    normal_cutoff = cutoff_freq / nyquist
    b, a = butter(order, normal_cutoff, btype='high', analog=False)
    filtered_signal = lfilter(b, a, signal)
    return filtered_signal

def low_pass_filter(signal, cutoff_freq, sample_rate, order=4):
    nyquist = 0.5 * sample_rate
    normal_cutoff = cutoff_freq / nyquist
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    filtered_signal = lfilter(b, a, signal)
    return filtered_signal
```

Datos_ECI_SGP4.py.

Me entrega los resultados del SGP4, IGRF y vector sol en ECI y los guarda en un .csv para su uso en otra función de Python con la parte de la dinámica de actitud

```
#%% Librerias a utilizar
from skyfield.api import utc
from datetime import datetime, timedelta
from sgp4.earth_gravity import wgs84
from sgp4.io import twoline2rv
import pyIGRF
import numpy as np
import functions

#%% Define la ubicación de tu archivo TLE y encuentra la condicion inicial del estado
tle_file = "suchai_3.txt"

# Lee el contenido del archivo
with open(tle_file, "r") as file:
    tle_lines = file.read().splitlines()

# Asegúrate de que el archivo contiene al menos dos líneas de TLE
if len(tle_lines) < 2:
    print("El archivo TLE debe contener al menos dos líneas.")
else:
    # Convierte las líneas del archivo en un objeto Satrec
    satellite = twoline2rv(tle_lines[0], tle_lines[1], wgs84)

    # Define la fecha inicial
    start_time = datetime(2023, 11, 1, 12, 0, 0)  # Ejemplo: 1 de noviembre de 2023, 12:00:00

    # Define el tiempo de propagación en segundos
    # propagation_time = 60*60*24*187
    propagation_time = 600000



    #posicion y velocidad del satelite en la fecha inicial
    position_i, velocity_i = satellite.propagate(start_time.year, start_time.month, start_time.day,
        start_time.hour, start_time.minute, start_time.second)

    #Para transformar el vector a LLA inicial
```

```
start_time_gps = datetime(2023, 11, 1, 12, 0, 0,tzinfo=utc)  # Ejemplo: 5 de julio de 2023, 12:00:00
lla_i = functions.eci2lla(position_i,start_time_gps)

#Obtener fuerzas magneticas de la Tierra inicial
Bi = pyIGRF.igrf_value(lla_i[0],lla_i[1],lla_i[2]/1000, start_time.year)
Bi_fn = np.array([Bi[3], Bi[4], Bi[5]])*1e-9
Bi_f = Bi_fn/np.linalg.norm(Bi_fn)

#obtener vector sol ECI inicial
jd2000i = functions.datetime_to_jd2000(start_time)
sunvectorin = functions.sun_vector(jd2000i)
sunvectori = sunvectorin / np.linalg.norm(sunvectorin)



#%% Listas donde guardar las variables ECI y body

positions = [position_i]
velocities = [velocity_i]
latitudes = [lla_i[0]]
longitudes = [lla_i[1]]
altitudes = [lla_i[2]]
Bx_IGRF = [Bi_f[0]]
By_IGRF = [Bi_f[1]]
Bz_IGRF = [Bi_f[2]]
vsun_x = [sunvectori[0]]
vsun_y = [sunvectori[1]]
vsun_z = [sunvectori[2]]



#%% Propagacion SGP4 y obtencion de vectores magneticos y sol en ECI a traves del tiempo

# Inicializa el tiempo actual un segundo despues del inicio
deltat = 2
current_time = start_time+ timedelta(seconds=deltat)
current_time_gps = start_time_gps + timedelta(seconds=deltat)

t0 = 0
t_aux = [t0]
```

```
while current_time < start_time + timedelta(seconds=propagation_time):
    t0 = t0 + deltat
    position, velocity = satellite.propagate(
        current_time.year, current_time.month, current_time.day,
        current_time.hour, current_time.minute, current_time.second
    )

    lla = functions.eci2lla(position,current_time_gps)

    #Obtener fuerzas magneticas de la Tierra
    Bm_PD = pyIGRF.igrf_value(lla[0],lla[1],lla[2]/1000, current_time.year)
    B_fn = np.array([Bm_PD[3], Bm_PD[4], Bm_PD[5]])
    B_f = B_fn / np.linalg.norm(B_fn)
    jd2000 = functions.datetime_to_jd2000(current_time)
    print(current_time)
    sunvector_n = functions.sun_vector(jd2000)
    sunvector = sunvector_n / np.linalg.norm(sunvector_n)

    t_aux.append(t0)
    positions.append(position)
    velocities.append(velocity)
    latitudes.append(lla[0])
    longitudes.append(lla[1])
    altitudes.append(lla[2])
    Bx_IGRF.append(B_f[0])
    By_IGRF.append(B_f[1])
    Bz_IGRF.append(B_f[2])
    vsun_x.append(sunvector[0])
    vsun_y.append(sunvector[1])
    vsun_z.append(sunvector[2])


    current_time += timedelta(seconds= deltat)
    current_time_gps += timedelta(seconds=deltat)

t_aux = np.array(t_aux)
positions = np.array(positions)
velocities = np.array(velocities)
latitudes = np.array(latitudes)
longitudes = np.array(longitudes)
```

```
    altitudes = np.array(altitudes)
    Bx_IGRF = np.array(Bx_IGRF)
    By_IGRF = np.array(By_IGRF)
    Bz_IGRF = np.array(Bz_IGRF)
    vsun_x = np.array(vsun_x)
    vsun_y = np.array(vsun_z)
    vsun_z = np.array(vsun_z)

    #%% Guardar datos en un archivo csv

    # Nombre del archivo
    archivo = "vectores_60k.csv"

    # Abrir el archivo en modo escritura
    with open(archivo, 'w') as f:
        # Escribir los encabezados
        f.write("t_aux, position_x,position_y,position_z, velocity_x, velocity_y, velocity_z, latitudes, longitudes, altitudes, Bx_IGRF, By_IGRF, Bz_IGRF, vsun_x, vsun_y, vsun_z\n")

        # Escribir los datos en filas
        for i in range(len(t_aux)):
            f.write("{},{},{},{},{}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}\n".format(
                t_aux[i], positions[i,0], positions[i,1],positions[i,2],velocities[i,0],velocities[i,1],velocities[i,2],
                latitudes[i],longitudes[i], altitudes[i], Bx_IGRF[i], By_IGRF[i],
                Bz_IGRF[i], vsun_x[i], vsun_y[i], vsun_z[i]
            ))

    print("Vectores guardados en el archivo:", archivo)
```

Memoria_control_aplicado_bad - copia.py.

Simulación para sensores de mala calidad (para los demás casos de sensores se cambia el error del cuaternion, y para el caso de los niveles de actuadores se considera un sensor de alta calidad con restricciones en las acciones de control en base a los niveles de componentes encontrados).

```
"""
#%% Librerias a utilizar

import numpy as np
from scipy.spatial.transform import Rotation
import functions
import pandas as pd



#%% importar archivo con SGP4 y mediciones ECI

# Nombre del archivo CSV
archivo_csv = "vectores_40k.csv"

# Leer el archivo CSV en un DataFrame de pandas
df = pd.read_csv(archivo_csv)

# Convertir el DataFrame a un array de NumPy
array_datos = df.to_numpy()

t_aux = array_datos[:,0]
Bx_IGRF = array_datos[:,10]
By_IGRF = array_datos[:,11]
Bz_IGRF = array_datos[:,12]
vsun_x = array_datos[:,13]
vsun_y = array_datos[:,14]
vsun_z = array_datos[:,15]

position = np.transpose(np.vstack((array_datos[:,1], array_datos[:,2], array_datos[:,3])))
velocity = np.transpose(np.vstack((array_datos[:,4], array_datos[:,5], array_datos[:,6])))

Z_orbits = position[:,:] / np.linalg.norm(position[:,:])
X_orbits = np.cross(velocity[:,:],Z_orbits) / np.linalg.norm(np.cross(velocity[:,:],Z_orbits))
Y_orbits = np.cross(Z_orbits,X_orbits)

q0_e2o = []
```

```
q1_e2o = []
q2_e2o = []
q3_e2o = []

Bx_orbit = []
By_orbit = []
Bz_orbit = []
vx_sun_orbit = []
vy_sun_orbit = []
vz_sun_orbit = []
for i in range(len(t_aux)):
    Rs_ECI_orbit = np.vstack((X_orbits[i,:],Y_orbits[i,:],Z_orbits[i,:]))
    q_ECI_orbit= Rotation.from_matrix(Rs_ECI_orbit).as_quat()

    q0_e2o.append(q_ECI_orbit[0])
    q1_e2o.append(q_ECI_orbit[1])
    q2_e2o.append(q_ECI_orbit[2])
    q3_e2o.append(q_ECI_orbit[3])

    B_ECI_quat = [Bx_IGRF[i],By_IGRF[i],Bz_IGRF[i],0]
    inv_q_e2o = functions.inv_q(q_ECI_orbit)
    B_orbit = functions.quat_mult(functions.quat_mult(q_ECI_orbit,B_ECI_quat),inv_q_e2o)
    B_orbit_n = np.array([B_orbit[0],B_orbit[1],B_orbit[2]])
    B_orbit = B_orbit_n / np.linalg.norm(B_orbit_n)

    vsun_ECI_quat = [vsun_x[0],vsun_y[0],vsun_z[0],0]
    inv_qi_s = functions.inv_q(q_ECI_orbit)
    vsun_orbit = functions.quat_mult(functions.quat_mult(q_ECI_orbit,vsun_ECI_quat),inv_qi_s)
    vsun_orbit_n = np.array([vsun_orbit[0],vsun_orbit[1],vsun_orbit[2]])
    vsun_orbit =  vsun_orbit_n / np.linalg.norm(vsun_orbit_n)

    Bx_orbit.append(B_orbit[0])
    By_orbit.append(B_orbit[1])
    Bz_orbit.append(B_orbit[2])
    vx_sun_orbit.append(vsun_orbit[0])
    vy_sun_orbit.append(vsun_orbit[1])
    vz_sun_orbit.append(vsun_orbit[2])

q0_e2o = np.array(q0_e2o)
q1_e2o = np.array(q1_e2o)
q2_e2o = np.array(q2_e2o)
```

```
q3_e2o = np.array(q3_e2o)
Bx_orbit =np.array(Bx_orbit)
By_orbit =np.array(By_orbit)
Bz_orbit =np.array(Bz_orbit)
vx_sun_orbit = np.array(vx_sun_orbit)
vy_sun_orbit = np.array(vy_sun_orbit)
vz_sun_orbit = np.array(vz_sun_orbit)

#%% q y w inicial real y q TRIAD

# Condiciones iniciales dadas del vector estado (cuaternion y velocidad angular)
q = np.array([-0.7071/np.sqrt(3),0.7071/np.sqrt(3),-0.7071/np.sqrt(3),0.7071])
w= np.array([0.0001,0.0001,0.0001])

#Obtener fuerzas magneticas de la Tierra inicial orbit y BODY
Bi_quat = [Bx_IGRF[0],By_IGRF[0],Bz_IGRF[0],0]
inv_qi_b = functions.inv_q(q)
Bi_body = functions.quat_mult(functions.quat_mult(q,Bi_quat),inv_qi_b)
Bi_body_n = np.array([Bi_body[0],Bi_body[1],Bi_body[2]])
Bi_body = Bi_body_n / np.linalg.norm(Bi_body_n)

#vector sol en orbit y BODY
vsuni_quat = [vsun_x[0],vsun_y[0],vsun_z[0],0]
inv_qi_s = functions.inv_q(q)
vsuni_body = functions.quat_mult(functions.quat_mult(q,vsuni_quat),inv_qi_s)
vsuni_body_n = np.array([vsuni_body[0],vsuni_body[1],vsuni_body[2]])
vsuni_body =  vsuni_body_n / np.linalg.norm(vsuni_body_n)

#Creacion de la solucion TRIAD como matriz de rotacion en orbit-body
DCM = functions.TRIAD(Bi_body,vsuni_body,Bi_quat[:3],vsuni_quat[:3])
q_TRIAD = Rotation.from_matrix(DCM).as_quat()

# Matriz de rotacion ECI2ORBIT
q_e2o = np.array([q0_e2o[0],q1_e2o[0],q2_e2o[0],q3_e2o[0]])
R_ECI_orbit = functions.quaternion_to_dcm(q_e2o)

# Matriz de rotacion ORBIT2BODY
R_orbit_B = np.dot(np.transpose(DCM),R_ECI_orbit)
q_orbit_B= Rotation.from_matrix(R_orbit_B).as_quat()

RPY_TRIAD = functions.quaternion_to_euler(q_TRIAD)
```

```
#%% Listas para guardar las variables de estado reales, TRIAD y body

I_x = 0.037
I_y = 0.036
I_z = 0.006

Bx_bodys = [Bi_body[0]]
By_bodys = [Bi_body[1]]
Bz_bodys = [Bi_body[2]]
vsun_bodys_x = [vsuni_body[0]]
vsun_bodys_y = [vsuni_body[1]]
vsun_bodys_z = [vsuni_body[2]]

q0_rot = [q_orbit_B[0]]
q1_rot = [q_orbit_B[1]]
q2_rot = [q_orbit_B[2]]
q3_rot = [q_orbit_B[3]]

q0_TRIADS = [q_TRIAD[0]]
q1_TRIADS = [q_TRIAD[1]]
q2_TRIADS = [q_TRIAD[2]]
q3_TRIADS = [q_TRIAD[3]]

w0_values = [w[0]]
w1_values = [w[1]]
w2_values = [w[2]]

#%% Control lineal PD

w0_o = 0.00163
w0_oi = np.array([w0_o,0,0])

w0_test = 0
w1_test = 0
w2_test = 0

A = functions.A_PD(I_x,I_y,I_z,w0_o, w0_test,w1_test,w2_test)
A= np.where(np.abs(A) == 0, 0, A)

B_body_ctrl = np.array([1617.3,4488.9,46595.8])*1e-9
```

```
B = functions.B_PD(I_x,I_y,I_z,B_body_ctrl)

# # quiza para magnetorquer medio
# Kp_x =  -0.0261
# Kp_y = 0.0027
# Kp_z = -0.6306
# Kd_x = -17.2535
# Kd_y = -10.6473
# Kd_z =  138.2978

# Kp_x =  -241.799276978197
# Kp_y = 43.3569691036660
# Kp_z = -3059.34072509731
# Kd_x = -560.886975412642
# Kd_y = -155.273483672191
# Kd_z =  -5161.73551572602

Kp_x =  -11.7126815151457
Kp_y = 0.0215395552140476
Kp_z = -1.94092971659118
Kd_x = 2.23100994690840
Kd_y = -0.0591645752827404
Kd_z =  -473.824574185555




K_Ex_app = functions.K(Kp_x, Kp_y, Kp_z, Kd_x, Kd_y, Kd_z)

K_Ps = K_Ex_app[:,:3]
K_Ds = K_Ex_app[:,3:6]

#%% sensores
# Características del magnetómetro bueno
rango = 800000  # nT
ruido = 0.1*1e-9  # nT/√Hz

#Caracteristicas del magnetometro malo
rango_bad = 75000 #nT
```

```
ruido_bad = 1.18*1e-9 #nT/√Hz

#caracteristicas del sun sensor malo
acc_bad = 5 #°
sigma_bad = functions.sigma_sensor(acc_bad)

#Caracteristicas del sun sensor intermedio
acc_med = 1 #°
sigma_med = functions.sigma_sensor(acc_med)

#caracteristicas del sun sensor bueno
sigma_good = 0.05

# Datos del giroscopio malo
bias_instability_bad = 0.05 / 3600 *np.pi/180 # <0.10°/h en radianes por segundo
noise_rms_bad = 0.12*np.pi/180 # 0.12 °/s en radianes por segundo

#Datos del giroscopio medio
bias_instability_med = 0.03 / 3600 * np.pi/180  # <0.06°/h en radianes por segundo
noise_rms_med = 0.050 *np.pi/180 # 0.12 °/s rms en radianes por segundo

#Datos del giroscopio bueno
bias_instability_good = 0 # <0.06°/h en radianes por segundo
noise_rms_good= 0.033 *np.pi/180  # 0.050 °/s rms en radianes por segundo

deltat = 2 #cambiable segun otro archivo .py
h = 0.1
#%% Propagacion del control PD

np.random.seed(42)

for i in range(len(t_aux)-1):
   q_TRIAD_PD   =   np.array([q0_TRIADS[-1],q1_TRIADS[-1],q2_TRIADS[-1],q3_TRIADS[-
1]])
   w_TRIAD_PD = np.array([w0_values[-1],w1_values[-1],w2_values[-1]])
   #w_TRIAD_PD = w_in_NL - np.dot(quaternion_to_dcm(q_TRIAD_PD),w0_oi)

   dx_PD_NL = np.hstack((np.transpose(q_TRIAD_PD[:3]), np.transpose(w_TRIAD_PD)))
   u_PD_NL = np.dot(K_Ex_app,dx_PD_NL)

   q_in_PD = np.array([q0_rot[-1],q1_rot[-1],q2_rot[-1],q3_rot[-1]])
```

```
x_PD_REAL = np.hstack((np.transpose(q_in_PD[:3]), np.transpose(w_TRIAD_PD)))
u_PD_REAL =np.dot(K_Ps,q_in_PD[:3]) + np.dot(K_Ds,w_TRIAD_PD)

for j in range(int(deltat/h)):
    q_rot,w_new = functions.rk4_step_PD(functions.dynamics, x_PD_REAL, A, B,u_PD_NL, h)
    if  1-q_rot[0]**2-q_rot[1]**2-q_rot[2]**2 < 0:
        q_rot = q_rot / np.linalg.norm(q_rot)
        dx_PD_NL = np.hstack((np.transpose(q_rot), np.transpose(w_new)))
        q3s_rot = 0

    else:
        dx_PD_NL = np.hstack((np.transpose(q_rot), np.transpose(w_new)))
        q3s_rot = np.sqrt(1-q_rot[0]**2-q_rot[1]**2-q_rot[2]**2)

q0_rot.append(q_rot[0])
q1_rot.append(q_rot[1])
q2_rot.append(q_rot[2])
q3_rot.append(q3s_rot)
q_rot_ind = np.array([q0_rot[-1],q1_rot[-1],q2_rot[-1],q3_rot[-1]])

w_gyros_bad = functions.simulate_gyros_reading(w_new,0, 0)
w0_values.append(w_gyros_bad[0])
w1_values.append(w_gyros_bad[1])
w2_values.append(w_gyros_bad[2])

B_quat_ECI = [Bx_IGRF[i+1],By_IGRF[i+1],Bz_IGRF[i+1],0]
B_quat = [Bx_orbit[i+1],By_orbit[i+1],Bz_orbit[i+1],0]
inv_q_b = functions.inv_q(q_rot_ind)
B_body = functions.quat_mult(functions.quat_mult(q_rot_ind,B_quat),inv_q_b)
B_body_n = np.array([B_body[0],B_body[1],B_body[2]])
B_body = B_body_n / np.linalg.norm(B_body_n)
B_magn_bad = functions.simulate_magnetometer_reading(B_body, ruido_bad)
Bx_bodys.append(B_magn_bad[0])
By_bodys.append(B_magn_bad[1])
Bz_bodys.append(B_magn_bad[2])

vsun_quat_ECI = [vsun_x[i+1],vsun_y[i+1],vsun_z[i+1],0]
vsun_quat = [vx_sun_orbit[i+1],vy_sun_orbit[i+1],vz_sun_orbit[i+1],0]
inv_q_s = functions.inv_q(q_rot_ind)
vsun_body = functions.quat_mult(functions.quat_mult(q_rot_ind,vsun_quat),inv_q_s)
vsun_body_n = np.array([vsun_body[0],vsun_body[1],vsun_body[2]])
```

```
    vsun_body = vsun_body_n / np.linalg.norm(vsun_body_n)
    vsun_bad = functions.simulate_sunsensor_reading(vsun_body, sigma_med)
    vsun_bodys_x.append(vsun_bad[0])
    vsun_bodys_y.append(vsun_bad[1])
    vsun_bodys_z.append(vsun_bad[2])

    delta_q = np.random.rand(4)*0.35#np.linalg.norm(np.array([0.1,ruido_bad,sigma_bad]))
    q_TRIADA_n = q_rot_ind + delta_q
    q_TRIADA = q_TRIADA_n / np.linalg.norm(q_TRIADA_n)

    # DCM_PD = functions.TRIAD(B_body,vsun_body,B_quat_ECI[:3],vsun_quat_ECI[:3])
    # q_TRIADAECI = Rotation.from_matrix(DCM_PD).as_quat()

    # # Matriz de rotacion ECI2ORBIT
    # q_eci2orbit = np.array([q0_e2o[i+1],q1_e2o[i+1],q2_e2o[i+1],q3_e2o[i+1]])
    # R_ECI2orbit = functions.quaternion_to_dcm(q_eci2orbit)

    # # Matriz de rotacion ORBIT2BODY
    # R_orbit2Body = np.dot(np.transpose(DCM_PD),R_ECI2orbit)
    # q_TRIADA = Rotation.from_matrix(R_orbit2Body).as_quat()

    q0_TRIADS.append(q_TRIADA[0])
    q1_TRIADS.append(q_TRIADA[1])
    q2_TRIADS.append(q_TRIADA[2])
    q3_TRIADS.append(q_TRIADA[3])

Bx_bodys = np.array(Bx_bodys)
By_bodys = np.array(By_bodys)
Bz_bodys = np.array(Bz_bodys)

vsun_bodys_x = np.array(vsun_bodys_x)
vsun_bodys_y = np.array(vsun_bodys_y)
vsun_bodys_z = np.array(vsun_bodys_z)

q0_TRIADS = np.array(q0_TRIADS)
q1_TRIADS = np.array(q1_TRIADS)
q2_TRIADS = np.array(q2_TRIADS)
q3_TRIADS = np.array(q3_TRIADS)

q0_rot = np.array(q0_rot)
q1_rot = np.array(q1_rot)
```

```
q2_rot = np.array(q2_rot)
q3_rot = np.array(q3_rot)

w0_values = np.array(w0_values)
w1_values = np.array(w1_values)
w2_values = np.array(w2_values)

q_k_all_id = np.vstack((q0_rot,q1_rot,q2_rot,q3_rot))
q_k_all_id_t = np.transpose(q_k_all_id)
RPY_all_id = []

for i in range(len(t_aux)):
    RPY_EKF_id = functions.quaternion_to_euler(q_k_all_id_t[i,:])
    RPY_all_id.append(RPY_EKF_id)

RPY_all_id = np.array(RPY_all_id)

#%% Guardar datos para graficar en otro codigo

# Nombre del archivo
archivo_c = "control_bad_o.csv"

# Abrir el archivo en modo escritura
with open(archivo_c, 'w') as f:
    # Escribir los encabezados
    f.write("t_aux, Roll,Pitch,Yaw, q0_rot, q1_rot, q2_rot, q3_rot, q0_TRIAD, q1_TRIAD, q2_TRIAD, q3_TRIAD, w0_values, w1_values, w2_values _z\n")

    # Escribir los datos en filas
    for i in range(len(t_aux)):
        f.write("{}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}, {}\n".format(
            t_aux[i], RPY_all_id[i,0], RPY_all_id[i,1],RPY_all_id[i,2],q0_rot[i],q1_rot[i],q2_rot[i],
            q3_rot[i],q0_TRIADS[i],q1_TRIADS[i], q2_TRIADS[i], q3_TRIADS[i],
            w0_values[i], w1_values[i], w2_values[i]
        ))

print("Vectores guardados en el archivo:", archivo_c)
```

Graficas_memoria.py.

Script de Python que permite seleccionar el .csv para graficar y obtener los resultados de MoP de apuntamiento.

```
#%% Librerias a utilizar
import matplotlib.pyplot as plt
import pandas as pd
import functions
from scipy.signal import welch
import numpy as np

# archivo_c_csv = "control.csv"
# archivo_c_csv = "control5s.csv"
# archivo_c_csv = "control_TRIAD.csv"
# archivo_c_csv = "control_bad_gyros.csv"
# archivo_c_csv = "control_orbit.csv"
# archivo_c_csv = "control_orbits.csv"

   # "control_good_magnet_o_Worbit.csv"

#%%

# Definir la lista de archivos CSV disponibles
archivos_disponibles = [
   "control_bad_o.csv",
   "control_med_o.csv",
   "control_good_magnet_o.csv",
   "control_good_magnetmed_o.csv",
   "control_good_magnetbad_o.csv",
   "control_good_magnetTRIAD_o.csv"

]

# Mostrar el menú al usuario
print("Seleccione un archivo CSV para abrir:")
for i, archivo in enumerate(archivos_disponibles, 1):
   print(f"{i}. {archivo}")

# Obtener la elección del usuario
opcion = int(input("Ingrese el número de archivo deseado: "))
```

```
# Validar la opción del usuario
if 1 <= opcion <= len(archivos_disponibles):
    archivo_c_csv = archivos_disponibles[opcion - 1]

    # Leer el archivo CSV en un DataFrame de pandas
    df_c = pd.read_csv(archivo_c_csv)

    # Convertir el DataFrame a un array de NumPy
    array_datos_c = df_c.to_numpy()

    # Ahora puedes usar 'array_datos_c' en tu código
    print(f"Archivo seleccionado: {archivo_c_csv}")
else:
    print("Opción inválida. Por favor, ingrese un número válido.")

print("\n")

t_aux = array_datos_c[:,0]
Roll = array_datos_c[:,1]
Pitch =  array_datos_c[:,2]
Yaw =  array_datos_c[:,3]
q0_rot = array_datos_c[:,4]
q1_rot = array_datos_c[:,5]
q2_rot = array_datos_c[:,6]
q3_rot = array_datos_c[:,7]
q0_TRIADS = array_datos_c[:,8]
q1_TRIADS = array_datos_c[:,9]
q2_TRIADS = array_datos_c[:,10]
q3_TRIADS = array_datos_c[:,11]
w0_values = array_datos_c[:,12]
w1_values = array_datos_c[:,13]
w2_values = array_datos_c[:,14]
# pot_act_x = abs(array_datos_c[:,15])
# pot_act_y = abs(array_datos_c[:,16])
# pot_act_z = abs(array_datos_c[:,17])
# pot_magn_x = array_datos_c[:,18]
# pot_magn_y = array_datos_c[:,19]
# pot_magn_z = array_datos_c[:,20]

#Filtro pasa alto para el Jitter y pasa bajo para exactitud de apuntamiento y agilidad
```

```
Roll_high_pass = functions.high_pass_filter(Roll, 10, len(t_aux))
Roll_low_pass = functions.low_pass_filter(Roll, 10, len(t_aux))
Pitch_high_pass = functions.high_pass_filter(Pitch, 10, len(t_aux))
Pitch_low_pass = functions.low_pass_filter(Pitch, 10, len(t_aux))
Yaw_high_pass = functions.high_pass_filter(Yaw, 10, len(t_aux))
Yaw_low_pass = functions.low_pass_filter(Yaw, 10, len(t_aux))

#Caso de ruido debido a valores de q3 forzados a 0
if archivo_c_csv == "control_good_magnetmed_o.csv": #or archivo_c_csv == "control_good_magnetbad_o.csv":
    # Definir la sección en el cual ocurre el fallo
    start_index = np.where(t_aux >= 45000)[0][0]
    end_index = np.where(t_aux <= 55000)[0][-1]

    Yaw_lowpass = functions.low_pass_filter(Yaw[start_index:end_index+1],5, len(t_aux[start_index:end_index+1]))
    Yaw_low_pass_section = functions.low_pass_filter(Yaw_high_pass[start_index:end_index+1], 5, len(t_aux[start_index:end_index+1]))
    Yaw_new = np.zeros_like(Yaw)
    Yaw_high_pass_new = np.zeros_like(Yaw)
    Yaw_new[int(t_aux[0]):start_index] = Yaw[int(t_aux[0]):start_index]
    Yaw_high_pass_new[int(t_aux[0]):start_index] = Yaw_high_pass[int(t_aux[0]):start_index]
    Yaw_new[start_index:end_index+1]= Yaw_lowpass
    Yaw_high_pass_new[start_index:end_index+1]= Yaw_low_pass_section
    Yaw_new[end_index:int(t_aux[-1])] = Yaw[end_index:int(t_aux[-1])]
    Yaw_high_pass_new[end_index:int(t_aux[-1])]= Yaw_high_pass[end_index:int(t_aux[-1])]
    Yaw = Yaw_new
    Yaw_high_pass = Yaw_high_pass_new

frequencies_R, psd_R = welch(Roll_high_pass, len(t_aux), nperseg=1024)
frequencies_P, psd_P = welch(Pitch_high_pass, len(t_aux), nperseg=1024)
frequencies_Y, psd_Y = welch(Yaw_high_pass, len(t_aux), nperseg=1024)

psd_R_R =[]
psd_P_R =[]
psd_Y_R =[]

for i in range(len(frequencies_R)):
    psd_R_r = np.real(psd_R[i])
    psd_P_r = np.real(psd_P[i])
    psd_Y_r = np.real(psd_Y[i])
```

```
    psd_R_R.append(psd_R_r)
    psd_P_R.append(psd_P_r)
    psd_Y_R.append(psd_Y_r)

psd_R_R = np.array(psd_R_R)
psd_P_R = np.array(psd_P_R)
psd_Y_R = np.array(psd_Y_R)

# Definir los anchos de banda deseados
bandwidth_1 = (0, 10000)  # Ancho de banda 1 en Hz

# Calcular la PSD dentro de los anchos de banda específicos
indices_bandwidth_1_R = np.where((frequencies_R >= bandwidth_1[0]) & (frequencies_R <= bandwidth_1[1]))
psd_bandwidth_1_R = np.trapz(psd_R[indices_bandwidth_1_R], frequencies_R[indices_bandwidth_1_R])

indices_bandwidth_1_P = np.where((frequencies_P >= bandwidth_1[0]) & (frequencies_P <= bandwidth_1[1]))
psd_bandwidth_1_P = np.trapz(psd_P[indices_bandwidth_1_P], frequencies_P[indices_bandwidth_1_P])

indices_bandwidth_1_Y = np.where((frequencies_Y >= bandwidth_1[0]) & (frequencies_Y <= bandwidth_1[1]))
psd_bandwidth_1_Y = np.trapz(psd_Y[indices_bandwidth_1_Y], frequencies_Y[indices_bandwidth_1_Y])

psd_RPY = np.array([psd_bandwidth_1_R,psd_bandwidth_1_P,psd_bandwidth_1_Y])
print("las densidades de potencia espectral en Roll, Pitch y Yaw son: \n",psd_RPY)
print("\n")


# Encontrar el tiempo de asentamiento en segundos de cada angulo de Euler
settling_band_R = 1
settling_band_P = 1
settling_band_Y = 1

settling_error_sup_R = np.full(len(t_aux),settling_band_R+ Roll_low_pass[-1])
settling_error_inf_R = np.full(len(t_aux),-settling_band_R+ Roll_low_pass[-1])

settling_error_sup_P = np.full(len(t_aux),settling_band_P+ Pitch_low_pass[-1])
```

```
settling_error_inf_P = np.full(len(t_aux),-settling_band_P+ Pitch_low_pass[-1])

settling_error_sup_Y = np.full(len(t_aux),settling_band_Y+ Yaw_low_pass[-1])
settling_error_inf_Y = np.full(len(t_aux),-settling_band_Y+ Yaw_low_pass[-1])



settling_time_indices_R = []
start_index_R = None

for i in range(len(Roll_low_pass)):
    if Roll_low_pass[i] <= settling_error_sup_R[i] and Roll_low_pass[i] >= settling_error_inf_R[i]:
        if start_index_R is None:
            start_index_R = i
    else:
        if start_index_R is not None:
            settling_time_indices_R.append((start_index_R, i - 1))
            start_index_R = None

if start_index_R is not None:
    settling_time_indices_R.append((start_index_R, len(Roll_low_pass) - 1))

if settling_time_indices_R:
    settling_times_R = []
    for start, end in settling_time_indices_R:
        settling_times_R.append((t_aux[start], t_aux[end]))
    print("Tiempos de asentamiento en Roll:")
    for start, end in settling_times_R:
        print("Inicio:", start,"[s]", "Fin:", end,"[s]")
else:
    print("La señal no entra en la banda de asentamiento.")

print("\n")
settling_time_indices_P = []
start_index_P = None

for i in range(len(Pitch_low_pass)):
    if Pitch_low_pass[i] <= settling_error_sup_P[i] and Pitch_low_pass[i] >=
settling_error_inf_P[i]:
        if start_index_P is None:
            start_index_P = i
    else:
```

```
        if start_index_P is not None:
            settling_time_indices_P.append((start_index_P, i - 1))
            start_index_P = None

if start_index_P is not None:
    settling_time_indices_P.append((start_index_P, len(Pitch_low_pass) - 1))

if settling_time_indices_P:
    settling_times_P = []
    for start, end in settling_time_indices_P:
        settling_times_P.append((t_aux[start], t_aux[end]))
    print("Tiempos de asentamiento en Pitch:")
    for start, end in settling_times_P:
        print("Inicio:", start,"[s]", "Fin:", end,"[s]")
else:
    print("La señal no entra en la banda de asentamiento.")

print("\n")
settling_time_indices_Y = []
start_index_Y = None

for i in range(len(Yaw_low_pass)):
    if Yaw_low_pass[i] <= settling_error_sup_Y[i] and Yaw_low_pass[i] >=
settling_error_inf_Y[i]:
        if start_index_Y is None:
            start_index_Y = i
    else:
        if start_index_Y is not None:
            settling_time_indices_Y.append((start_index_Y, i - 1))
            start_index_Y = None

if start_index_Y is not None:
    settling_time_indices_Y.append((start_index_Y, len(Yaw_low_pass) - 1))

if settling_time_indices_Y:
    settling_times_Y = []
    for start, end in settling_time_indices_Y:
        settling_times_Y.append((t_aux[start], t_aux[end]))
    print("Tiempos de asentamiento en Yaw:")
    for start, end in settling_times_Y:
        print("Inicio:", start,"[s]", "Fin:", end,"[s]")
```

```
else:
    print("La señal no entra en la banda de asentamiento.")

print("\n")


#Seccion para calcular la exactitud de apuntamiento en cada angulo de Euler
time_R = np.array(settling_times_R[-1])
data_R = Roll_low_pass[int(time_R[0]/2):int(time_R[1]/2)]
accuracy_R = abs(np.mean(data_R))

time_P = np.array(settling_times_P[-1])
data_P = Pitch_low_pass[int(time_P[0]/2):int(time_P[1]/2)]
accuracy_P = abs(np.mean(data_P))

time_Y = np.array(settling_times_Y[-1])
data_Y = Yaw_low_pass[int(time_Y[0]/2):int(time_Y[1]/2)]
accuracy_Y = abs(np.mean(data_Y))

accuracy_RPY = np.array([accuracy_R,accuracy_P,accuracy_Y])
print("La exactitud de apuntamiento para Roll, Pitch y Yaw son respecticamente: \n", accuracy_RPY, "[°]")


#%%

plt.figure(figsize=(12, 6))
plt.plot(t_aux, Roll, label='Roll')
plt.plot(t_aux, Pitch, label='Pitch')
plt.plot(t_aux, Yaw, label='Yaw')
plt.xlabel('Tiempo [s]')
plt.ylabel('Ángulos de Euler [°]')
plt.legend()
plt.title('Obtencion de los ángulos de Euler')
# plt.xlim(0.8e7,1.7e7)
# plt.ylim(-15,2)
plt.grid()
plt.show()
# plt.set_yli1m(-10, 10)  # Ajusta los límites en el eje Y
```

```
# fig0, axes0 = plt.subplots(nrows=1, ncols=2, figsize=(14, 4))

# axes0[0].plot(t_aux, yaw_b, label='Yaw nivel bajo')
# axes0[0].plot(t_aux, yaw_m, label='Yaw nivel medio')
# axes0[0].plot(t_aux, yaw_g, label='Yaw nivel alto')
# axes0[0].set_xlabel('Tiempo [s]')
# axes0[0].set_ylabel('Ángulos de Euler [°]')
# axes0[0].legend()
# axes0[0].grid()
# #axes0[0].set_ylim(-1, 1)  # Ajusta los límites en el eje Y

# axes0[1].plot(t_aux, yaw_b, label='Yaw nivel bajo')
# axes0[1].plot(t_aux, yaw_m, label='Yaw nivel medio')
# axes0[1].plot(t_aux, yaw_g, label='Yaw nivel alto')
# axes0[1].set_xlabel('Tiempo [s]')
# axes0[1].set_ylabel('Ángulos de Euler [°]')
# # axes0[1].legend()
# axes0[1].grid()
# axes0[1].set_ylim(-20, -5)  # Ajusta los límites en el eje Y
# axes0[1].set_xlim(150000, 400000)  # Ajusta los límites en el eje Y

# plt.tight_layout()
# plt.show()


fig0, axes0 = plt.subplots(nrows=1, ncols=3, figsize=(16, 4))

axes0[0].plot(t_aux, q0_rot, label='q0 PD_NL')
axes0[0].plot(t_aux, q1_rot, label='q1 PD_NL')
axes0[0].plot(t_aux, q2_rot, label='q2 PD_NL')
axes0[0].plot(t_aux, q3_rot, label='q3 PD_NL')
axes0[0].set_xlabel('Tiempo [s]')
axes0[0].set_ylabel('cuaternión [-]')
axes0[0].legend()
axes0[0].set_title('cuaterniones controlados llevados a 000')
axes0[0].grid()
#axes0[0].set_ylim(-1, 1)  # Ajusta los límites en el eje Y

axes0[1].plot(t_aux, w0_values, label='w0 PD_NL')
axes0[1].plot(t_aux, w1_values, label='w1 PD_NL')
axes0[1].plot(t_aux, w2_values, label='w2 PD_NL')
```

```
axes0[1].set_xlabel('Tiempo [s]')
axes0[1].set_ylabel('velocidad angular [rad/s]')
axes0[1].legend()
axes0[1].set_title('velocidades angulares controlados llevados a 000')
axes0[1].grid()


axes0[2].plot(t_aux, q0_TRIADS, label='q0 TRIAD')
axes0[2].plot(t_aux, q1_TRIADS, label='q1 TRIAD')
axes0[2].plot(t_aux, q2_TRIADS, label='q2 TRIAD')
axes0[2].plot(t_aux, q3_TRIADS, label='q3 TRIAD')
axes0[2].set_xlabel('Tiempo [s]')
axes0[2].set_ylabel('cuaternión TRIAD [-]')
axes0[2].legend()
axes0[2].set_title('cuaterniones TRIAD controlados llevados a 000')
axes0[2].grid()

plt.tight_layout()
plt.show()


# plt.figure(figsize=(12, 6))
# plt.plot(t_aux, Roll_high_pass, label='Roll')
# plt.plot(t_aux, Pitch_high_pass, label='Pitch')
# plt.plot(t_aux, Yaw_high_pass, label='Yaw')
# plt.xlabel('Tiempo[s]')
# plt.ylabel('Ángulos de Euler [°]')
# plt.legend()
# plt.title('Filtros pasa alto')
# # plt.xlim(0.8e7,1.7e7)
# # plt.ylim(-0.002,0.002)
# plt.grid()
# plt.show()

# plt.figure(figsize=(12, 6))
# plt.plot(t_aux, Roll_low_pass, label='Roll')
# plt.plot(t_aux, Pitch_low_pass, label='Pitch')
# plt.plot(t_aux, Yaw_low_pass, label='Yaw')
# plt.xlabel('Tiempo[s]')
# plt.ylabel('Ángulos de Euler [°]')
# plt.legend()
```

```
# plt.title('Filtros pasa bajo')
# # plt.xlim(0.8e7,1.7e7)
# # plt.ylim(-0.002,0.002)
# plt.grid()
# plt.show()



# fig0, axes0 = plt.subplots(nrows=2, ncols=1, figsize=(7, 7))

# axes0[0].plot(t_aux, Roll_low_pass, label='Roll')
# axes0[0].set_xlabel('Tiempo [s]')
# axes0[0].set_ylabel('Ángulos de Euler [°]')
# axes0[0].legend()
# axes0[0].grid()
# #axes0[0].set_ylim(-1, 1)  # Ajusta los límites en el eje Y

# axes0[1].plot(t_aux, Roll_low_pass, label='Roll')
# axes0[1].plot(t_aux, settling_error_sup_R , label='SET_sup_roll')
# axes0[1].plot(t_aux, settling_error_inf_R , label='SET_inf_roll')
# axes0[1].set_xlabel('Tiempo [s]')
# axes0[1].set_ylabel('Ángulos de Euler [°]')
# axes0[1].legend()
# # axes0[1].set_title('Filtros pasa bajo')
# axes0[1].grid()
# axes0[1].set_ylim(-25,0)  # Ajusta los límites en el eje Y

# plt.tight_layout()
# plt.show()

# fig0, axes0 = plt.subplots(nrows=2, ncols=1, figsize=(7, 7))

# axes0[0].plot(t_aux, Pitch_low_pass, label='Pitch')
# axes0[0].set_xlabel('Tiempo [s]')
# axes0[0].set_ylabel('Ángulos de Euler [°]')
# axes0[0].legend()
# axes0[0].grid()
# #axes0[0].set_ylim(-1, 1)  # Ajusta los límites en el eje Y

# axes0[1].plot(t_aux, Pitch_low_pass, label='Pitch')
# axes0[1].plot(t_aux, settling_error_sup_P , label='SET_sup_pitch')
# axes0[1].plot(t_aux, settling_error_inf_P , label='SET_inf_pitch')
```

```
# axes0[1].set_xlabel('Tiempo [s]')
# axes0[1].set_ylabel('Ángulos de Euler [°]')
# axes0[1].legend()
# # axes0[1].set_title('Filtros pasa bajo')
# axes0[1].grid()
# axes0[1].set_ylim(-10,10)  # Ajusta los límites en el eje Y

# plt.tight_layout()
# plt.show()

# fig0, axes0 = plt.subplots(nrows=2, ncols=1, figsize=(7, 7))

# axes0[0].plot(t_aux, Yaw_low_pass, label='Yaw')
# axes0[0].set_xlabel('Tiempo [s]')
# axes0[0].set_ylabel('Ángulos de Euler [°]')
# axes0[0].legend()
# axes0[0].grid()
# #axes0[0].set_ylim(-1, 1)  # Ajusta los límites en el eje Y

# axes0[1].plot(t_aux, Yaw_low_pass, label='Yaw')
# axes0[1].plot(t_aux, settling_error_sup_Y , label='SET_sup_yaw')
# axes0[1].plot(t_aux, settling_error_inf_Y , label='SET_inf_yaw')
# axes0[1].set_xlabel('Tiempo [s]')
# axes0[1].set_ylabel('Ángulos de Euler [°]')
# axes0[1].legend()
# # axes0[1].set_title('Filtros pasa bajo')
# axes0[1].grid()
# axes0[1].set_ylim(-25,0)  # Ajusta los límites en el eje Y

# plt.tight_layout()
# plt.show()

# plt.figure(figsize=(12, 6))
# plt.plot(t_aux, Roll_low_pass, label='Roll')
# plt.plot(t_aux, settling_error_sup_R , label='SET_sup_roll')
# plt.plot(t_aux, settling_error_inf_R , label='SET_inf_roll')
# plt.xlabel('Tiempo[s]')
# plt.ylabel('Ángulos de Euler [°]')
# plt.legend()
# plt.title('Filtros pasa bajo')
# # plt.ylim(-25,0)
```

```
# # plt.xlim(0.8e7,1.7e7)
# # plt.ylim(-0.002,0.002)
# plt.grid()
# plt.show()

# plt.figure(figsize=(12, 6))
# plt.plot(t_aux, Pitch_low_pass, label='Pitch')
# plt.plot(t_aux, settling_error_sup_P , label='SET_sup_pitch')
# plt.plot(t_aux, settling_error_inf_P , label='SET_inf_pitch')
# plt.xlabel('Tiempo[s]')
# plt.ylabel('Ángulos de Euler [°]')
# plt.legend()
# plt.title('Filtros pasa bajo')
# # plt.ylim(-10,10)
# # plt.xlim(0.8e7,1.7e7)
# # plt.ylim(-0.002,0.002)
# plt.grid()
# plt.show()

# plt.figure(figsize=(12, 6))
# plt.plot(t_aux, Yaw_low_pass, label='Yaw')
# plt.plot(t_aux, settling_error_sup_Y , label='SET_sup_yaw')
# plt.plot(t_aux, settling_error_inf_Y , label='SET_inf_yaw')
# plt.xlabel('Tiempo[s]')
# plt.ylabel('Ángulos de Euler [°]')
# plt.legend()
# plt.title('Filtros pasa bajo')
# # plt.ylim(-25,0)
# # plt.xlim(0.8e7,1.7e7)
# # plt.ylim(-0.002,0.002)
# plt.grid()
# plt.show()



# plt.tight_layout()
# plt.show()
# plt.figure(figsize=(12, 6))
# plt.plot(t_aux,  pot_magn_x, label='potencia gastada en x')
# plt.plot(t_aux,  pot_magn_y, label='potencia gastada en y')
# plt.plot(t_aux,  pot_magn_z, label='potencia gastada en z')
# plt.xlabel('Tiempo')
```

```
# plt.ylabel('Potencia [W]')
# plt.legend()
# plt.title('Potencia gastada por el sensor en sus tres direcciones')
# # plt.ylim(-2,2)
# # plt.xlim(0.8e7,1.7e7)
# # plt.ylim(-0.002,0.002)
# plt.grid()
# plt.show()

# plt.figure(figsize=(12, 6))
# plt.plot(t_aux,  pot_act_x, label='potencia gastada en x')
# plt.plot(t_aux,  pot_act_y, label='potencia gastada en y')
# plt.plot(t_aux,  pot_act_z, label='potencia gastada en z')
# plt.xlabel('Tiempo')
# plt.ylabel('Potencia [W]')
# plt.legend()
# plt.title('Potencia gastada por el actuador en sus tres direcciones')
# # plt.ylim(-2,2)
# # plt.xlim(0.8e7,1.7e7)
# # plt.ylim(-0.002,0.002)
# plt.grid()
# plt.show()

# plt.figure(figsize=(10, 6))
# plt.semilogy(frequencies_R, psd_R_R, label='PSD Roll')
#   plt.fill_between(frequencies_R[indices_bandwidth_1_R],   psd_R_R[indices_bandwidth_1_R],
alpha=0.3, label='Ancho de banda 1')
# plt.title('Densidad Espectral de Potencia con Anchos de Banda Específicos en Roll')
# plt.xlabel('Frecuencia (Hz)')
# plt.ylabel('Densidad Espectral de Potencia')
# plt.legend()
# plt.grid(True)
# plt.show()

# plt.figure(figsize=(10, 6))
# plt.semilogy(frequencies_P, psd_P_R, label='PSD Pitch')
#   plt.fill_between(frequencies_P[indices_bandwidth_1_P],   psd_P_R[indices_bandwidth_1_P],
alpha=0.3, label='Ancho de banda 1')
# plt.title('Densidad Espectral de Potencia con Anchos de Banda Específicos en Pitch')
# plt.xlabel('Frecuencia (Hz)')
# plt.ylabel('Densidad Espectral de Potencia')
```

```
# plt.legend()
# plt.grid(True)
# plt.show()



# plt.figure(figsize=(10, 6))
# plt.semilogy(frequencies_Y, psd_Y_R, label='PSD Yaw')
#   plt.fill_between(frequencies_Y[indices_bandwidth_1_Y],   psd_Y_R[indices_bandwidth_1_Y],
alpha=0.3, label='Ancho de banda 1')
# plt.title('Densidad Espectral de Potencia con Anchos de Banda Específicos en Yaw')
# plt.xlabel('Frecuencia (Hz)')
# plt.ylabel('Densidad Espectral de Potencia')
# plt.legend()
# plt.grid(True)
# plt.show()
```

render.py.

Al utilizar algún .csv de la simulación, me genera una animación 3D del Cubesat cambiando su posición respecto a la tierra y su orientación a través del tiempo utilizando raylib.py

```
# pip install raylib-py
import raylibpy as rp
import sys
import os
from ctypes import byref
import matplotlib.pyplot as plt
import pandas as pd
import functions
from scipy.signal import welch
import numpy as np


#%%

# Nombre del archivo CSV
archivo_csv = "vectores_60k.csv"

# Leer el archivo CSV en un DataFrame de pandas
df = pd.read_csv(archivo_csv)

# Convertir el DataFrame a un array de NumPy
array_datos = df.to_numpy()

position = np.transpose(np.vstack((array_datos[:,1], array_datos[:,2], array_datos[:,3])))
velocity = np.transpose(np.vstack((array_datos[:,4], array_datos[:,5], array_datos[:,6])))


# Definir la lista de archivos CSV disponibles
archivos_disponibles = [
    "control_bad_o.csv",
    "control_med_o.csv",
    "control_good_magnet_o.csv",
    "control_good_magnetmed_o.csv",
    "control_good_magnetbad_o.csv",
    "control_good_magnetbad_o60k.csv"
]
```

```
# Mostrar el menú al usuario
print("Seleccione un archivo CSV para abrir:")
for i, archivo in enumerate(archivos_disponibles, 1):
    print(f"{i}. {archivo}")

# Obtener la elección del usuario
opcion = int(input("Ingrese el número de archivo deseado: "))

# Validar la opción del usuario
if 1 <= opcion <= len(archivos_disponibles):
    archivo_c_csv = archivos_disponibles[opcion - 1]

    # Leer el archivo CSV en un DataFrame de pandas
    df_c = pd.read_csv(archivo_c_csv)

    # Convertir el DataFrame a un array de NumPy
    array_datos_c = df_c.to_numpy()

    # Ahora puedes usar 'array_datos_c' en tu código
    print(f"Archivo seleccionado: {archivo_c_csv}")
else:
    print("Opción inválida. Por favor, ingrese un número válido.")

print("\n")

t_aux = array_datos_c[:,0]
q0_rot = array_datos_c[:,4]
q1_rot = array_datos_c[:,5]
q2_rot = array_datos_c[:,6]
q3_rot = array_datos_c[:,7]



#%%

GLSL_VERSION = 330

def main():

    screenWidth = 1200
    screenHeight = 700
```

```
rp.init_window(screenWidth, screenHeight, "raylib [shaders] example - simple shader mask")

camera = rp.Camera()
#camera.position  = rp.Vector3(0.0, 1.0, 2.0)
# camera.position  = rp.Vector3(10, 10, 10)
# camera.target  = rp.Vector3(0.0, 0.0, 0.0)
camera.up  = rp.Vector3(0.0, 1.0, 0.0)
camera.fovy  = 45.0
camera.projection  = rp.CAMERA_PERSPECTIVE

# torus = rp.gen_mesh_torus(0.3, 1, 16, 32)
# model1 = rp.load_model_from_mesh(torus)
cube = rp.gen_mesh_cube(0.25, 0.25, 0.5)
model1 = rp.load_model_from_mesh(cube)
sphere = rp.gen_mesh_sphere(6.371, 16, 16)
model2 = rp.load_model_from_mesh(sphere)

texDiffuse = rp.load_texture("resources/plasma.png")
texDiffuse2 = rp.load_texture("resources/earth albedo.png")
model1.materials[0].maps[rp.MATERIAL_MAP_ALBEDO].texture  = texDiffuse
model2.materials[0].maps[rp.MATERIAL_MAP_ALBEDO].texture  = texDiffuse2

background = rp.load_texture("resources/space3.png")

framesCounter = 0

rp.disable_cursor()

rp.set_target_fps(60)

i = 0
default_font = rp.get_font_default()


while not rp.window_should_close():
    rp.update_camera(camera.byref, rp.CAMERA_FIRST_PERSON)
    framesCounter += 1
    Q = rp.Quaternion(q0_rot[i], q1_rot[i], q2_rot[i], q3_rot[i])
```

```
    # rp.set_shader_value(shader, shaderFrame, byref(rp.Int(framesCounter)), rp.SHADER_UNIFORM_INT)
    model1.transform = rp.quaternion_to_matrix(Q)
    with rp.drawing():
        rp.clear_background(rp.BLUE)
        rp.draw_texture(background, 0, 0, rp.WHITE)

        with rp.mode3d(camera):
            camera.position = rp.Vector3(position[i][0]/1000 - 5, position[i][1]/1000+ 8, position[i][2]/1000- 5)
            camera.target = rp.Vector3(position[i][0]/1000, position[i][1]/1000, position[i][2]/1000)

            rp.draw_model(model1, rp.Vector3(position[i][0]/1000, position[i][1]/1000, position[i][2]/1000), 1, rp.WHITE)
            rp.draw_model(model2, rp.Vector3(0, 0, 0), 1, rp.WHITE)
            #rp.draw_model_ex(model2, rp.Vector3(-0.5, 0.0, 0.0), Vector3(1.0, 1.0, 0.0), 50, Vector3(1.0, 1.0, 1.0), WHITE)
            # draw_model(model3, Vector3(0.0, 0.0, -1.5), 1, WHITE)

            rp.draw_grid(10, 1.0)

        # end_mode3d()

        # rp.draw_rectangle(16, 698, rp.measure_text(f"Frame: {framesCounter}", 20) + 8, 42, rp.BLUE)
        # rp.draw_text(f"Frame: {framesCounter}", 20, 700, 20, rp.WHITE)

        rp.draw_fps(10, 10)
        #rp.draw_text(str(Q),10,10,12,rp.WHITE)
        #rp.draw_text(str(t_aux[i]),10,10,12,rp.WHITE)
        rp.draw_text_ex(default_font,"El tiempo es: " + str(t_aux[i]) + "[s]",rp.Vector2(10, 40),12,5,rp.WHITE)
    # end_drawing()
    i = i+1
    if i > len(q0_rot):
        i = 0

  rp.unload_model(model1)
  # rp.unload_model(model2)
  # rp.unload_model(model3)
```

```
    rp.unload_texture(texDiffuse)
    # rp.unload_texture(texMask)

    # rp.unload_shader(shader)

    rp.close_window()

    return 0


if __name__ == "__main__":
    if len(sys.argv) >= 2 and isinstance(sys.argv[1], str):
        os.chdir(sys.argv[1])
    print("Working dir:", os.getcwd())
    sys.exit(main())
```