DISEÑO Y DESARROLLO DE UN ENTORNO HARDWARE IN THE LOOP SIMULATION PARA PROBAR UN SISTEMA DE CONTROL PARA UN COHETE A ESCALA

Universidad
del Cauca

ANGELA LUCIA MUTIZ POPAYAN
STELLA MARIA SCARPETTA PIZO

UNIVERSIDAD DEL CAUCA
FACULTAD DE CIENCIAS NATURALES, EXACTAS Y DE LA EDUCACIÓN
DEPARTAMENTO DE FÍSICA
POPAYAN
2023

DISEÑO Y DESARROLLO DE UN ENTORNO HARDWARE IN THE LOOP SIMULATION PARA PROBAR UN SISTEMA DE CONTROL PARA UN COHETE A ESCALA

TRABAJO DE GRADO PARA OPTAR EL TITULO DE
INGENIERAS FÍSICAS

ANGELA LUCIA MUTIZ POPAYAN
STELLA MARIA SCARPETTA PIZO

DIRECTOR:
Ing. MARIO ANDRÉS CÓRDOBA

UNIVERSIDAD DEL CAUCA
FACULTAD DE CIENCIAS NATURALES, EXACTAS Y DE LA EDUCACIÓN
DEPARTAMENTO DE FÍSICA
POPAYAN
2023

Nota de Aceptación

Director - Ing. Mario Andres Cordoba

Jurado - Mag. Lina Marcela Jaller

Jurado - Dr. Diego Fernando Coral

Popayán-Cauca, 18 de Octubre de 2023

A nuestras
familias
por su amor y
dedicación en este
arduo camino.

Agradecimientos

A medida que avanzábamos en esta investigación, nuestras familias y amigos más cercanos nos brindaron palabras de aliento impregnadas de amor y comprensión. Agradecemos sinceramente su apoyo incondicional, el cual fue fundamental para no desistir en los momentos de mayor dificultad en el transcurso de este proyecto.

Queremos manifestar nuestro más sincero agradecimiento al Ingeniero Mario Córdoba, nuestro director, por su valiosa asesoría y enseñanzas que han sido fundamentales en nuestro desarrollo académico. Igualmente, deseamos expresar nuestra profunda gratitud a nuestros respetables docentes y profesores, quienes a lo largo de nuestra formación nos han proporcionado el conocimiento y las herramientas necesarias para alcanzar este punto crucial en nuestras vidas como futuras profesionales. Su dedicación y enseñanzas han sido pilares esenciales en nuestra preparación y crecimiento académico. A todos ellos, les estamos sinceramente agradecidas.

Contenido

Lísta de imágenes

Pág

Lísta de Tablas

Pág

Lista de Símbolos

a	Ángulo de Ataque
a	Aceleración
g_0	Aceleración de la Gravedad
A_e	Área efectiva
A	Área del Cono
S	Área de la Seccion Transversal
CP	Centro de Presiones
CG	Centro de Gravedad
K_P	Constante Proporcional
K_D	Constante Derivativa
C_D	Coeficiente Dinámico
C_{eff}	Coeficiente de Velocidad Efectiva del Propelente
ρ	Densidad del Aire
e_τ	Entrada del Controlador
T	Empuje
F_D	Fuerza de Arrastre
$F_{N\alpha}$	Fuerza de Normal
F	Fuerza
F_u	Fuerzas Externas
D	Fuerza de Rozamiento
G	Fuerza de Gravedad
F_m	Fuerza de Empuje Generada por el Cambio del Momentum
K	Ganancia Proporcional
p	Impulso
I_{sp}	Impulso Especifico
M	Masa Inicial
M_f	Masa Final
M	Momento de Giro
P_{mom}	Momentum Lineal
M_e	Masa Propelente
λ	Parametro Especifico
P_a	Presión Atmosférica
P_e	Presión

P_0	Presión de flujo
T_d	Tiempo Derivativo
T_i	Tiempo Integral
t	Tiempo
V	Velocidad
V	Velocidad del Cohete
c	Velocidad de escape
v_e	Velocidad de los Gases de escape

Resumen

La presente investigación se enfoca en el desarrollo e implementación de un sistema Hardware In The Loop Simulation (HILS) destinado a un cohete a escala. Esta técnica, ampliamente reconocida en el ámbito de la ingeniería, tiene como finalidad validar algoritmos de control mediante la ejecución en un procesador específico. Esto se logra mediante la creación de un entorno virtual que reproduce el comportamiento del sistema físico del cohete a escala y su interacción con el sistema de control, realizando un papel fundamental al permitir la evaluación del desempeño de los algoritmos antes de proceder a la construcción del prototipo físico, asegurando su correcto funcionamiento. Para optimizar la operación del sistema, se proyecta hacia la máxima simplicidad y aplicabilidad posible. El estudio se inicia con una introducción al microcontrolador elegido y al sistema a implementar. Posteriormente, se presentan detalladamente las ecuaciones de movimiento del cohete, considerando variables como el medio atmosférico terrestre, el comportamiento gravitacional y el sistema de referencia inercial del centro de masas terrestre. Además, se proporciona una breve contextualización sobre las clasificaciones existentes de los cohetes. Una vez establecido el modelo matemático y definido el sistema, se procede a seleccionar un cohete a escala de dimensiones reducidas que cumpla con los requisitos del estudio. Las masas y etapas del cohete se determinan en función de los parámetros proporcionados por Rocket Lab, para luego simular su comportamiento de vuelo mediante la utilización del sistema HILS. Finalmente, se realizan tanto simulaciones convencionales como de HILS para el Electron Rocket, obteniendo trayectorias detalladas y analizando la evolución de cinco parámetros críticos durante el vuelo del cohete: altitud, velocidad, masa, ángulos de Euler y trayectoria elipsoidal.

Capítulo 1

Introducción

En las últimas décadas, ha habido un gran interés en el desarrollo de técnicas y metodologías para mejorar el rendimiento y la seguridad de los sistemas de control de cohetes. En este campo, se han llevado a cabo numerosas investigaciones previas que han contribuido significativamente a nuestro entendimiento y desarrollo de sistemas de control de cohetes más eficientes y seguros. Entre las técnicas utilizadas en la industria aeroespacial destaca la simulación Hardware-In-The-Loop (HIL), la cual permite probar y validar el hardware de un sistema en un entorno controlado antes de su uso en una misión real. Con la simulación HIL se pueden probar la electrónica de control de vuelo, sensores y otros sistemas en un entorno simulado que imita las condiciones de un vuelo real. La técnica de simulación HIL se utiliza en múltiples campos de la ingeniería para proporcionar soluciones más efectivas a los problemas. En este contexto, la revisión bibliográfica se enfoca en investigaciones que involucran simulación en bucle y/o sistemas de control en cohetes, con el objetivo específico de respaldar los objetivos de este estudio.

Unas de las fuentes de referencia que se consultaron para comprender la implementación HILS, fueron "HILS Of Auto Take Off System: For High Speed UAV Using Rooster Rocket"[8] y .[A] Hardware-In-The-Loop Simulation Method For The Evaluation Of Flight Control Systems"[22].

En el estudio de Irwanto [8], se detalla el desarrollo de un sistema de piloto automático de alta velocidad, conocido como RKX-200TJ, iniciado en 2013. Este proyecto se distingue por la incorporación de una FPGA para optimizar el control y simplificar la gestión de entradas y salidas simultáneas. Asimismo, se activa el sistema de simulación Hardware-In-The-Loop (HILS) para poner en funcionamiento la Unidad de Medición Inercial (IMU), ubicada en la parte superior del hexápodo, con el fin de ilustrar la programación de circuito cerrado (PID) relacionada con la actitud del vehículo. Por otro lado, en el trabajo de Huang, Zhu & Zhao [22] también emplearon la simulación HILS para evaluar el rendimiento de los sistemas de control de vuelo (FCS), logrando una notable mejora en el desempeño del sistema de control. Ambas investigaciones subrayan la importancia y efectividad de la simulación Hardware-In-The-Loop (HILS)

en el desarrollo y mejora de sistemas de control, aportando avances significativos en términos de rendimiento y optimización.

Sin embargo, a pesar de estos avances en el campo de la simulación HILS para vehículos aéreos, aún existe una necesidad crítica de investigar el comportamiento y rendimiento de los sistemas en entornos complejos y altamente dinámicos. Específicamente, se necesita una mayor investigación sobre el comportamiento de cohetes durante la fase de ascenso y la fase de descenso, para mejorar la seguridad y la eficiencia de los mismos.

En este trabajo, se presenta una investigación que tiene como objetivo utilizar un enfoque de simulación HILS para probar y validar el rendimiento de un sistema de control de cohetes a escala. Los resultados obtenidos se discutirán detalladamente en el resto de la tesis.

1.1. Justificación del Trabajo.

En el transcurso de los años, la simulación HILS ha demostrado ser muy útil en la industria aeroespacial debido a los siguientes aportes:

1. **Pruebas más realistas:** El uso de un sistema HILS permite probar los sistemas aeroespaciales de forma más realista y precisa, esto se debe a que se puede simular un entorno similar al de un vuelo real.

2. **Ahorro de costos:** La simulación permite a los ingenieros detectar y corregir problemas antes de la implementación del sistema en una misión real, reduciendo significativamente los costos de producción y tiempo asociados con la detección de problemas en etapas tardías del proceso de desarrollo.

3. **Mayor eficiencia:** La simulación permite probar los sistemas aeroespaciales más rápidamente y con mayor eficiencia que las pruebas físicas en tierra o en vuelo, esto se debe a que se pueden realizar pruebas repetitivas y automatizadas en un entorno controlado y seguro.

4. **Desarrollo de sistemas de control:** La simulación se utiliza comúnmente para desarrollar, validar sistemas de control de vuelo; estos sistemas son importantes para garantizar el rendimiento y la seguridad del vehículo.

1.2. Descripción del Problema e Impacto.

En este contexto, el desafío que se enfrenta radica en la urgente necesidad de encontrar soluciones eficientes que permitan acelerar el proceso de desarrollo, reducir los costos asociados y mitigar los riesgos inherentes a las pruebas de nuevos diseños

de Unidades de Control Electrónico (ECU) destinados a cohetes reales, especialmente en el ámbito de la educación superior. Estas soluciones están diseñadas para facilitar pruebas realistas en futuros proyectos aeroespaciales, aprovechando la incorporación de microprocesadores de fácil acceso. Además, se busca que la interfaz sea altamente didáctica y que ofrezca a los estudiantes la posibilidad de probar sus propios diseños en entornos reales.

Dicho problema se deriva de una serie de desafíos:

- Elevados costos:Las pruebas realizadas en cohetes reales representan un desembolso económico considerable, que abarca desde los costos asociados a la adquisición de los cohetes hasta los componentes necesarios para las ECUs, además de los recursos requeridos para la ejecución y supervisión de estas pruebas.
- Complejidad inherente: La ejecución de pruebas en cohetes reales conlleva la coordinación de múltiples elementos y la gestión de sistemas altamente complejos. Cualquier fallo durante este proceso puede resultar costoso y demandar ajustes en el hardware.
- Riesgos latentes: Las pruebas en cohetes reales comportan riesgos inherentes, dado que un fallo en las ECUs puede conllevar a lanzamientos de cohetes fallidos o incluso desencadenar accidentes de considerable gravedad.
- Inversión de tiempo sustancial: Las pruebas en cohetes reales suelen extenderse por un prolongado período, lo que dilata el proceso de desarrollo y la implementación de mejoras en las ECUs

Por otro lado, la implantación de la simulación Hardware-in-the-Loop (HILS) para cohetes presenta un conjunto de ventajas y elementos novedosos en contraposición a las pruebas tradicionales realizadas en cohetes reales. Este enfoque despierta nuevas perspectivas al validar y optimizar los en entornos virtuales antes de su implementación en cohetes reales. Además, no solo arroja beneficios de índole técnica y financiera, sino que también promete incidir en el desarrollo y la formación de futuros ingenieros especializados en hardware y sistemas de control, ampliando el horizonte de posibilidades en el campo de las ciencias aeroespaciales. La investigación persigue la exploración y validación de la efectividad de la implementación de pruebas HILS, haciendo uso de un microcontrolador específico, como el microprocesador de Arduino Mega 2560 y datos reales como es el Rocket Lab, una empresa aeroespacial estadounidense con una filial de propiedad total en Nueva Zelanda, las ventajas significativas en términos de precisión, validación, reducción de riesgos y ahorro de costos en el desarrollo y prueba de ECUs para cohetes y sistemas aeroespaciales ademas como una herramienta fundamental para impulsar el avance en la tecnología espacial y la formación de profesionales altamente capacitados en esta área.

1.3. Objetivos

1.3.1. Objetivo General

Diseñar y desarrollar un entorno Hardware In The Loop Simulation (HILS) para probar un sistema de control específico para un cohete a escala.

1.3.2. Objetivos Específicos

1. Implementar el modelo matemático de la mecánica (dinámica y estática) de vuelo de un cohete a escala para un sistema de control específico.
2. Determinar las respectivas interfaces necesarias para la simulación con ayuda de un microprocesador.
3. Integrar el sistema HILS con el sistema de control desarrollado para la verificación en la simulación virtual.

Capítulo 2

Marco Teórico

La Tecnología HIL (Hardware-in-the-Loop), ha emergido como una herramienta fundamental en la industria, mostrando un rápido avance en su desarrollo. Se ha convertido en una técnica altamente efectiva para la prueba y aplicación de electrónica de potencia. Con HIL, los ingenieros y diseñadores cuentan con la capacidad de evaluar el rendimiento del controlador en tiempo real en conjunto con el sistema, lo que representa un recurso invaluable.

En este contexto, Arduino, es una plataforma de desarrollo de código abierto, emerge como una herramienta de notoria importancia. Fundamentada en microcontroladores, su versatilidad, amplia disponibilidad y una comunidad activa de desarrolladores y expertos la colocan en un plano destacado para la creación de sistemas embebidos.

La sinergia entre la Tecnología HIL y Arduino promete ventajas sustanciales. Facilita la creación de sistemas electrónicos de alta integridad al permitir la prueba en condiciones análogas a la realidad y proporcionar un entorno de desarrollo accesible y escalable. Esta convergencia no solo optimiza el proceso de diseño y validación, sino que también promueve la implementación efectiva y confiable de sistemas electrónicos de potencia.

2.1. Microcontrolador: Placa Arduino Mega 2560

El Arduino Mega 2560, basado en el Atmega2560, dispone de 54 pines digitales de entrada/salida, de los cuales 15 tienen la capacidad de operar como salidas PWM. Además, cuenta con 16 entradas analógicas, 4 UARTs (puertos serie hardware), un cristal oscilador de 16 MHz, una conexión USB, un puerto de alimentación, un encabezado ICSP y un botón de reinicio. Todo lo necesario para respaldar el funcionamiento del microcontrolador está integrado en la placa. Basta con conectarla a una computadora mediante un cable USB o alimentarla con un adaptador de corriente o batería para comenzar.

La Figura 2.1 muestra la Placa Arduino Mega 2560.

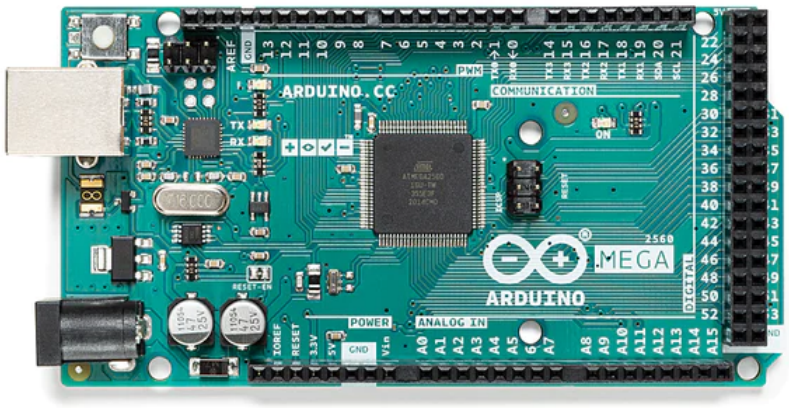


Figura 2.1: Placa Arduino Mega 2560 [3].

2.2. Hardware In The Loop Simulation (HILS)

La simulación Hardware-in-the-Loop (HILS) se ha establecido como el estándar preeminente para el desarrollo y prueba de sistemas de control, protección y monitoreo de alta complejidad. La implementación de HILS aborda de manera directa dos desafíos fundamentales que afectan el proceso de desarrollo de productos en diversas industrias: la reducción del tiempo de comercialización y la gestión de la complejidad del sistema. Históricamente, las pruebas de sistemas de control se han llevado a cabo en el campo, ya sea directamente en equipos físicos o en bancos de pruebas de potencia en laboratorios. Sin embargo, esta práctica puede resultar costosa, ineficiente y potencialmente peligrosa. Por ello, las pruebas HILS ofrecen una alternativa altamente efectiva y segura a los métodos tradicionales.

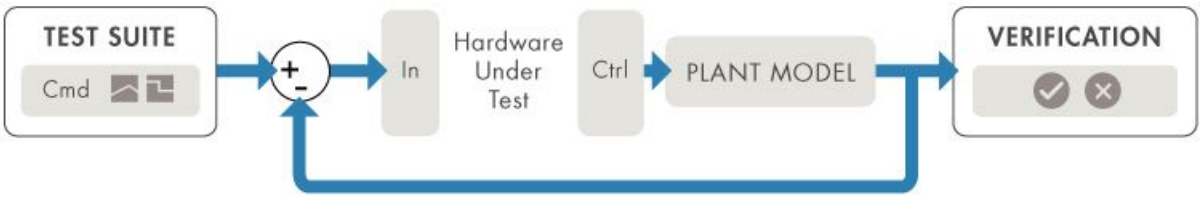


Figura 2.2: Diagrama Hardware In The Loop Simulation (HILS) [9]

El diagrama de bloques muestra una simulación HILS en la que el hardware bajo prueba es un controlador integrado y el modelo de planta es una representación de un sistema físico.

En la ejecución del sistema HILS, la infraestructura física se sustituye por un modelo virtual completamente equivalente. Este modelo opera en un simulador meticu-

losamente diseñado y equipado con entradas y salidas (E/S) capaces de interactuar de manera completa con sistemas de control y otros componentes relacionados. En esta configuración, el sistema HILS es capaz de emular con alta precisión la dinámica de la planta, incorporando tanto sensores como actuadores. Esto facilita la realización de pruebas integrales en circuito cerrado sin la necesidad de llevar a cabo pruebas físicas en sistemas reales.

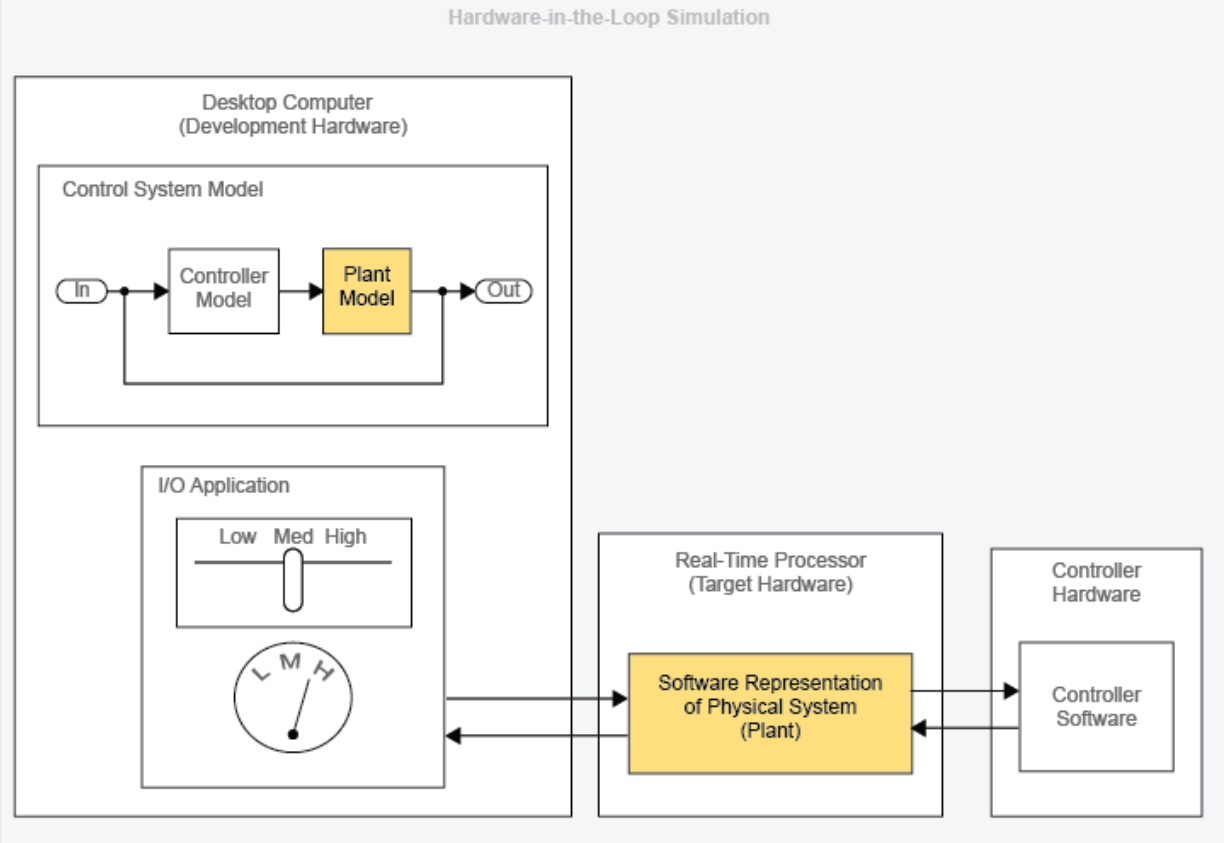


Figura 2.3: Hardware In The Loop Simulation [9]

Además, el sistema Hardware-in-the-Loop Simulation (HILS) no solo ofrece estas funcionalidades, sino que también atenúa de forma significativa las limitaciones que han caracterizado a los métodos de prueba conservativos a lo largo de los años. Al reducir los riesgos inherentes, los costos asociados y el tiempo global requerido para evaluar sistemas integrados de elevada complejidad, el sistema HILS ha alcanzado un estatus preeminente, erigiéndose como el estándar indiscutible en un variado espectro de industrias.[2].

Conforme avanza la tecnología, se evidencia una tendencia hacia la reducción de tamaño y el incremento de eficiencia en los componentes y sistemas electrónicos. Esta tendencia también prevalece en el ámbito de los sistemas espaciales, incluyendo los cohetes a escala. En este contexto, el sistema Hardware In The Loop Simulation (HILS)

desempeña un papel de vital importancia en el proceso de diseño. Ciertos componentes de hardware se integran en el bucle de simulación y operan a partir de los datos generados por un ordenador en funcionamiento. Esto conlleva a una diversidad de operaciones en comparación con el sistema de control.

Para desarrollar un sistema Hardware-in-the-Loop Simulation (HILS) eficaz, es imperativo tener en cuenta los principios fundamentales del sistema en cuestión. En el caso de un cohete, es esencial comprender en detalle su comportamiento durante el vuelo, así como considerar aspectos de diseño como el tipo de propelente a utilizar (ya sea líquido, sólido o una combinación de ambos) y el número de etapas que componen el cohete.[5]

En este contexto, los microcontroladores se erigen como un componente esencial para alcanzar los objetivos deseados en el diseño y prueba de sistemas HILS para cohetes. El microcontrolador es el responsable de dirigir el movimiento de información entre la placa base y la memoria del dispositivo,teniendo en cuenta lo anterior, es posible ejecutar mediante un lenguaje de programación los aspectos mencionados. Existen diversos softwares capaces de implementar el desarrollo del sistema HILS permitiendo verificar el sistema de control para aeronaves. Esta implementación permite realizar la respectiva adecuación e implementación de realidad virtual esperada, ofreciendo al usuario pueda detectar posibles fallas.

2.3. Modelo Matemático

Los cohetes son esenciales para la exploración espacial y la investigación científica. Comprender y predecir el comportamiento de un cohete es crucial para el éxito de las misiones espaciales y la optimización de los diseños de cohetes. Este modelo utiliza ecuaciones matemáticas para describir el vuelo, la propulsión y la variación de la masa del cohete debido al consumo de combustible.

2.3.1. Ecuaciones de Movimiento de un Cohete

Las ecuaciones de movimiento de un cohete describen su comportamiento durante el vuelo y tienen en cuenta las fuerzas que actúan sobre él, como la fuerza de empuje, la gravedad y la resistencia del aire. Las ecuaciones fundamentales incluyen la Segunda Ley de Newton, que relaciona la fuerza neta con la masa inercial y la aceleración del cohete; la ecuación de empuje, que depende de la velocidad de expulsión del propulsor y la tasa de cambio de masa; la fuerza gravitatoria, que se opone al movimiento ascendente; y la fuerza de arrastre, que depende de la forma del cohete y su velocidad. Estas ecuaciones se utilizan en conjunto para modelar el movimiento del cohete a lo largo del tiempo y calcular su posición, velocidad y aceleración durante el vuelo.[14]

Para obtener la ecuación, se parte de la definición del impulso en un sistema y su relación con la segunda ley de Newton. El impulso p se define como la integral de la velocidad V con respecto a la masa M:

$$p = \int_M V dM \tag{2.1}$$

Esta expresión nos da el impulso total en el sistema y es fundamental para comprender el cambio de cantidad de movimiento en función de la variación de la masa y la velocidad.

Ahora bien, al derivar el impulso con respecto al tiempo $\frac{\mathrm{d}p}{\mathrm{d}t}$, obtenemos la masa multiplicada por la aceleración (Ma), que es igual a la fuerza resultante F_u. Esto se basa en la segunda ley de Newton, que establece que la fuerza es igual a la masa multiplicada por la aceleración:

$$\frac{\mathrm{d}p}{\mathrm{d}t} = Ma = F_u \tag{2.2}$$

Siendo esenciales para el análisis de sistemas donde la masa y la velocidad son variables clave.

Teniendo en cuenta la relación del impulso en función del cambio de velocidad y masa, ecuación 2.2.Y dado que el impulso es el producto de la masa y la velocidad, $p = MV$, podemos derivar con respecto al tiempo para obtener:

$$\frac{dp}{dt} = M\frac{dV}{dt} + V\frac{dM}{dt} \tag{2.3}$$

Ahora, considerando un escenario donde la masa varía con el tiempo, $\frac{dM}{dt} \neq 0$, se puede reescribir la ecuación como:

$$F_u = M\frac{dV}{dt} + V\frac{dM}{dt} \tag{2.4}$$

Esta ecuación representa la fuerza resultante en un sistema donde tanto la masa como la velocidad están cambiando.

Luego, para considerar el cambio de impulso en un intervalo de tiempo dt, se puede multiplicar ambos lados por dt:

$$dp = MdV + VdM \tag{2.5}$$

Con esta expresión, se puede comprender cómo el cambio en la masa y la velocidad contribuye al cambio en el impulso.

$$dp = (M + dM)(V + dV) - udM - MV \tag{2.6}$$

Una vez teniendo esta expresión se puede llegar a la ecuación de movimiento de un cohete, con fuerzas externas F_u y una velocidad de escape c.

Expandiendo y simplificando la ecuación dp:

$$\begin{aligned} dp &= M(V + dV) + dM(V + dV) - udM - MV \\ dp &= MV + MdV + dMV + dMdV - udM - MV \\ dp &= MdV + dMV + dMdV - udM \end{aligned} \tag{2.7}$$

Reorganizando los términos y dividiendo por dt:

$$\begin{aligned} dp &= MdV + dMdV + dMV - udM \\ \frac{dp}{dt} &= M\frac{dV}{dt} + \frac{dM}{dt}V + \frac{dM}{dt}\frac{dV}{dt} - u\frac{dM}{dt} \end{aligned} \tag{2.8}$$

Dado que $m = -\frac{dM}{dt}$ y $c = V - u$, podemos reescribir la ecuación como:

$$\begin{aligned} \frac{dp}{dt} &= M\frac{dV}{dt} - mV - m\frac{dV}{dt} + um \\ \frac{dp}{dt} &= M\frac{dV}{dt} - mV - m\frac{dV}{dt} + um \\ \frac{dp}{dt} &= M\frac{dV}{dt} - m\frac{dV}{dt} - mV + um \\ \frac{dp}{dt} &= (M - m)\frac{dV}{dt} - m(V - u) \end{aligned} \tag{2.9}$$

Dado que $\frac{dp}{dt} = \sum F_u$:

$$\sum F_u = (M - m)\frac{dV}{dt} - m(V - u) \tag{2.10}$$

Finalmente, se obtiene la ecuación de movimiento del cohete:

$$M\frac{dV}{dt} = mc + \sum F_u \tag{2.11}$$

La ecuación de movimiento del cohete es la representación dinámica completa del comportamiento del cohete en vuelo, tomando en cuenta el empuje (T), la resistencia aerodinámica (D) y la pérdida de peso debida a la gravedad (Mg).

$$M\frac{\mathrm{d}V}{\mathrm{d}t} = T - D - Mg \tag{2.12}$$

Esta ecuación establece que la tasa de cambio de momento del cohete está determinada por la diferencia entre el empuje total (T), la resistencia aerodinámica (D) y la pérdida de peso debido a la gravedad (Mg). El término T se calcula como la suma del impulso producido por la expulsión de propulsante (mc) y la diferencia de presiones

en la tobera del motor ($A_e(p_e - p_0)$), que se simplifica como el producto de la variación de masa y la velocidad de escape específica (mc_{eff}).

Ahora considerando el parámetro del impulso específico I_{sp}, que se define como el impulso total T generado por el motor del cohete por unidad de masa consumida, dividido por la aceleración de la gravedad g_0. De igual manera, esta relación también puede expresarse como la velocidad de escape específica efectiva c_{eff} dividida por la aceleración de la gravedad g_0.

La ecuación para el impulso específico es:

$$I_{sp} = \frac{T}{mg_0} = \frac{c_{eff}}{g_0} \tag{2.13}$$

Donde:

- I_{sp}: Impulso específico del motor del cohete.
- T: Empuje total generado por el motor del cohete.
- m: Masa del cohete.
- g_0: Aceleración de la gravedad en la superficie de la Tierra.
- c_{eff}: Velocidad de escape específica efectiva.

Esta ecuación proporciona una medida de la eficiencia del motor del cohete en términos de cuánto impulso puede generar por unidad de masa consumida. Un I_{sp} alto indica un motor más eficiente.

2.3.2. Ecuación de Cohete de Tsiolkovski

Tsiolkovski definió que los vehículos de transporte espacial adquieren la energía necesaria para su movimiento a partir de un sistema de motores cohete, los cuales permiten el desarrollo de grandes velocidades con la transformación de la energía química de sus propelentes en energía cinética de gases de escape mediante el direccionamiento en un ducto propulsivo (tobera). La expulsión de una cantidad significativa de masa en poco tiempo (flujo másico) es lo que hace posible que los cohetes generen altas velocidades, acompañadas de fuerzas de empuje, pero limitados a causa del poco tiempo que dura la combustión[17].

La ecuación que relaciona la masa y la velocidad del sistema con su cantidad de movimiento lineal es:

$$P_{mom} = mV \tag{2.14}$$

donde P_{mom} es el momentum lineal del sistema, m la masa y V la velocidad.

Derivando esta ecuación respecto al tiempo se obtiene:

$$F_m = \frac{\mathrm{d}P}{\mathrm{d}t} = \frac{\mathrm{d}m}{\mathrm{d}t} v_e \tag{2.15}$$

dando como resultado que F_m es la fuerza de empuje generada por el cambio del momentum del cohete respecto al tiempo, o el producto de la variación de masas $\frac{\mathrm{d}m}{\mathrm{d}t}$ por la velocidad de salida de los gases respecto al vehículo representada por v_e. En otras palabras, el movimiento del cohete en una dimensión, sin considerar fuerzas externas (es decir, un cohete ideal) y en función de la fuerza de empuje se puede expresar como:

$$F_m = \frac{\mathrm{d}m}{\mathrm{d}t} v_e = \dot{m} v_e = \dot{m} g_0 I_{sp} \tag{2.16}$$

siendo $\dot{m}$ el flujo másico de los gases (tasa de salida de material eyectado por unidad de tiempo), $I_{sp} = \frac{v_e}{g_0}$ el impulso específico de los mismos y g_0 la aceleración de la gravedad en la superficie de la Tierra.

El cambio de velocidad del cohete en el vació depende entonces de la velocidad de expulsión de los gases y la perdida de masa respecto al tiempo:

$$dV = -v_e \frac{\mathrm{d}m}{\mathrm{m}} \tag{2.17}$$

La dirección de los vectores velocidad del cohete V son opuestos al vector velocidad de los gases de escape en la tobera v_e.

Integrando la ecuación 2.17 en limites de la masa inicial del cohete m_i y la masa final m_f, para velocidades correspondientes a V_i y V_f, tenemos:

$$\int_{v_i}^{v_f} dV = -v_e \int_{m_i}^{m_f} \frac{\mathrm{d}m}{\mathrm{m}} \tag{2.18}$$

Se obtiene:

$$\Delta V = V_f - V_i \tag{2.19}$$

$$V_f = V_i + v_e \ln \frac{\mathrm{m_i}}{\mathrm{m_f}} \tag{2.20}$$

donde la velocidad final V_f depende de la fracción de masa consumida, o la relación entre la masa inicial y la masa final[17].

2.3.3. Trayectoria de Lanzamiento

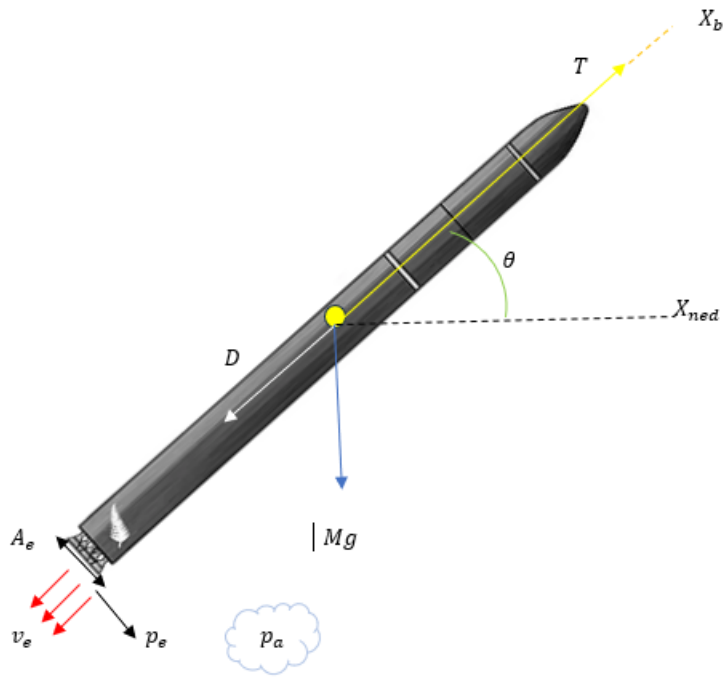


Figura 2.4: Diagrama de Lanzamiento [14]

La trayectoria de lanzamiento se puede dividir en tres fases distintas. La primera fase se produce mientras el cohete quema combustible y acelera hasta que agota su suministro de combustible. Luego, en la segunda fase, el cohete sigue una trayectoria balística sin propulsión adicional. Durante esta etapa, el cohete continúa ganando altitud hasta alcanzar su punto más alto, donde su velocidad vertical es cero. Finalmente, en la tercera fase, el cohete inicia su descenso. Para describir este escenario, se supone un campo gravitatorio uniforme y un vacío consistente[14].

En este contexto, se pueden derivar las ecuaciones de movimiento en las direcciones horizontal (x) y vertical (z) de la siguiente manera:

$$M\frac{\mathrm{d}V_x}{\mathrm{d}t} = T\cos\theta \tag{2.21}$$

$$M\frac{\mathrm{d}V_z}{\mathrm{d}t} = T\sin\theta - Mg_0 \tag{2.22}$$

2.3.4. Trayectorias de Ascenso Vertical

Estas trayectorias se distinguen por un movimiento vertical, en el cual el cohete se eleva de forma rectilínea hacia el firmamento. Durante esta fase, los motores del cohete proporcionan el impulso necesario al expulsar el propulsante a alta velocidad en dirección opuesta, generando así la fuerza esencial para contrarrestar la atracción gravitatoria. La velocidad de ascenso, la altitud alcanzada y la orientación del cohete están sujetas a la influencia de una amplia gama de factores. Es común que los cohetes dispongan de sistemas de control y guía con el propósito de asegurar un ascenso estable. En ciertas misiones, se procede a la separación de etapas durante esta fase con el fin de optimizar la eficiencia del lanzamiento. Las trayectorias de ascenso vertical representan un elemento crítico para la realización exitosa de lanzamientos de cohetes, especialmente en misiones que requieren alcanzar órbitas específicas o explorar el espacio más allá de la atmósfera terrestre.[14]

Este tipo de trayectorias es especialmente relevante para sondas y cohetes espaciales. En la práctica, estos cohetes suelen ser lanzados con un ángulo ligeramente diferente a los 90 grados, por ejemplo, a unos 80 grados.

Vuelo Vertical con Resistencia Aerodinámica

Durante el ascenso vertical a través de la atmósfera, la ecuación de movimiento se expresa de la siguiente manera:

$$M\frac{\mathrm{d}V}{\mathrm{d}t} = T - D - Mg_0 \quad \text{con} \quad D = C_D\frac{1}{2}\rho V^2 S \tag{2.23}$$

La integración numérica es una herramienta útil para calcular la trayectoria del cohete y evaluar las pérdidas debidas a la resistencia del aire. Durante el vuelo vertical propulsado, la velocidad (V) aumenta, mientras que la densidad del aire (ρ) disminuye con la altitud. Inicialmente, la velocidad tiene un papel dominante, pero a altitudes más elevadas, la densidad disminuye considerablemente hasta aproximarse a cero. A cierta altitud, la resistencia alcanza su punto máximo y luego comienza a disminuir. Es evidente que la presión dinámica (q) alcanza un valor máximo a una altitud de alrededor de 10 kilómetros. A altitudes por encima de los 60 kilómetros, la presión dinámica (q) es tan baja que, a pesar de la alta velocidad, la resistencia aerodinámica puede considerarse insignificante.[14]

Las pérdidas de velocidad, expresadas como ΔV_g y ΔV_D, están asociadas a dos aspectos fundamentales del vuelo del cohete. ΔV_g refleja la disminución de velocidad causada por la influencia gravitatoria durante el ascenso, mientras que ΔV_D representa la pérdida de velocidad debida a la resistencia del aire y otros factores de arrastre a medida que el cohete atraviesa la atmósfera. Ambas pérdidas están directamente ligadas a un parámetro crucial denominado índice de carga de empuje (ψ_0), calculado como la relación entre la fuerza de empuje inicial (T_0) y el producto de la masa inicial

(M_0) y la aceleración gravitatoria estándar (g_0).

Cuando el valor de ψ_0 aumenta, el tiempo de combustión (t_b) disminuye, lo que resulta en una reducción de ΔV_g. Por otro lado, un incremento en ψ_0 implica una mayor velocidad en las capas más densas de la atmósfera, lo que se traduce en un aumento de ΔV_D. Cabe destacar que, a diferencia del vuelo vertical en el vacío, la altitud máxima alcanzada no aumenta de manera lineal con ψ_0, sino que alcanza un valor máximo para un determinado valor de este parámetro.

2.4. Clasificación General de Cohetes

A lo largo de los años, el sector aeroespacial y la tecnología de cohetes han experimentado un notable progreso a gran escala. Por tanto, resulta pertinente llevar a cabo un análisis detenido de diversos aspectos fundamentales relacionados con las características físicas y el rendimiento global de los cohetes. En líneas generales, la clasificación de los cohetes se fundamenta en factores tales como sus aplicaciones específicas, el tipo de propulsor empleado y sus dimensiones físicas. Además, en ciertos casos, se presta atención a su modalidad de construcción y a los sistemas utilizados para el suministro del combustible. Con estas consideraciones en mente, se presenta la siguiente clasificación [18].

Aplicabilidad

A partir del cambio de milenio, se ha observado un significativo progreso en la industria espacial, caracterizado por la concepción de nuevos sistemas de lanzamiento, la expansión de organismos gubernamentales dedicados a la exploración espacial y la entrada de empresas privadas en el sector, impulsadas por avances tecnológicos de relevancia. Este crecimiento en el ámbito espacial ha dado origen al concepto de recuperación y reutilización de sistemas de lanzamiento.

La implementación exitosa del concepto de recuperación y reutilización de sistemas de lanzamiento fue liderada por SpaceX. Este logro se tradujo en una significativa reducción del costo por kilogramo de carga útil, alcanzando una disminución de hasta un 80 %. Esta proeza otorgó a SpaceX una ventaja competitiva considerable, llevándola al punto en que, a principios de 2020, la empresa pudo lanzar más de 61 toneladas de carga a órbita, superando en cantidad a la suma de lanzamientos realizados por las agencias espaciales chinas, europeas y rusas en conjunto [20].

A continuación, se enumeran algunas de las aplicaciones más destacadas de los cohetes:

- Satélites artificiales.

- Cohetes meteorológicos o sonda.
- Sistema auxiliar de propulsión para aviones en el despegue.

Motor

Es una de las partes mas importantes que compone un cohete, este es el que nos permite obtener el impulso necesario para elevarlo, existen diferentes tipos de motores para modelos espaciales, cabe destacar que, al combustible del motor en esta parte de la industria se le llama propelente, este tipo de combustible no necesita aire atmosférico para hacer funcionar el motor.

- **Propelente Solido:** Hay dos tipos de motores de propelente solido, los llamados convencionales o de pólvora y los de Composite.
- **Propelente Liquido:** Poseen una cámara de combustión separa del propelente.
- **Propelente Híbrido:** A este tipo de propelente se le debe depositar gas de Oxido de Nitrógeno para que actué como oxidante.

Tamaño

El tamaño de los cohetes depende principalmente de:

- Tiempo de la combustión.
- Impulso.
- Empuje dado.
- Peso del vehículo.
- Relación empuje-peso del sistema.

2.4.1. Tipos de Cohetes Sonda

El presente trabajo de investigación se enfocará en el estudio de cohetes de tipo sonda, diseñados con un propósito específico: la obtención de mediciones. Estos cohetes se componen esencialmente de dos componentes principales: un motor sólido y una carga útil. En muchos de estos programas, los motores son suministrados por entidades militares, lo que contribuye a mantener los costos en un rango reducido. La carga útil alberga los instrumentos encargados de llevar a cabo el experimento y transmitir los datos de vuelta a través de un radar instalado en la superficie terrestre [4].

A continuación, se presentan algunas clasificaciones y comparaciones de cohetes que actualmente se encuentran en la industria.

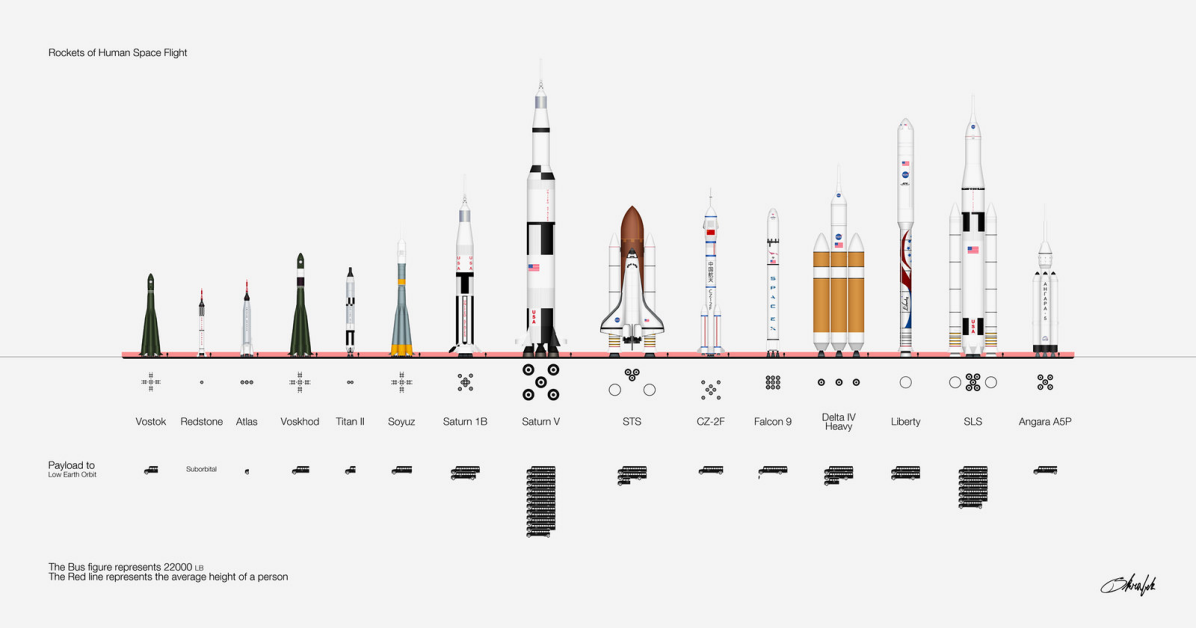


Figura 2.5: Comparación Entre Cohetes de Carga Útil[21]

En la figura 2.6, podemos evaluar la capacidad de cada uno de los cohetes en términos de su carga útil en una órbita sincrónica con el Sol, que corresponde a la órbita terrestre baja. Estas órbitas garantizan que el satélite pase sobre cada punto de la Tierra a la misma hora solar media todos los días. Los costos o presupuestos de desarrollo de estos cohetes todavía se encuentran en fase de estimación, y la mayoría de ellos están en proceso de pruebas y desarrollo.

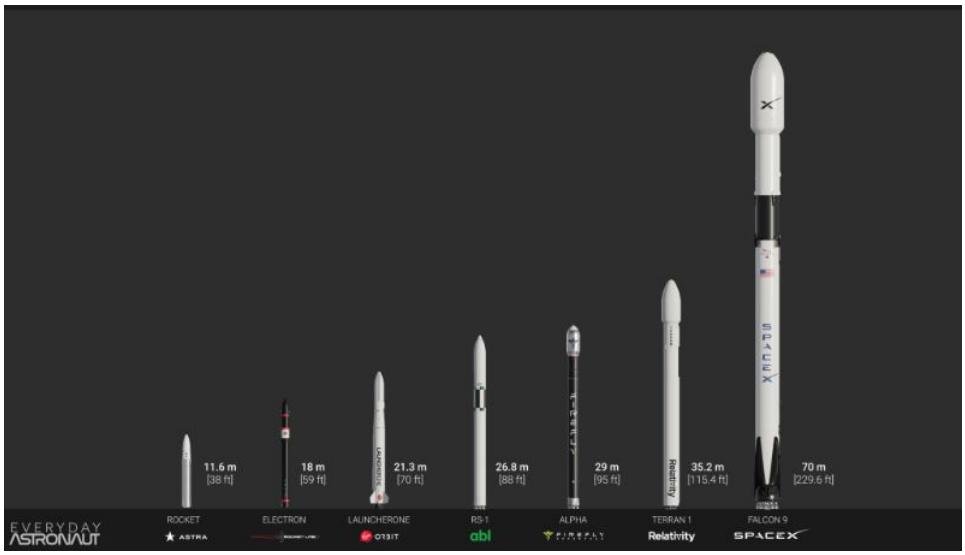
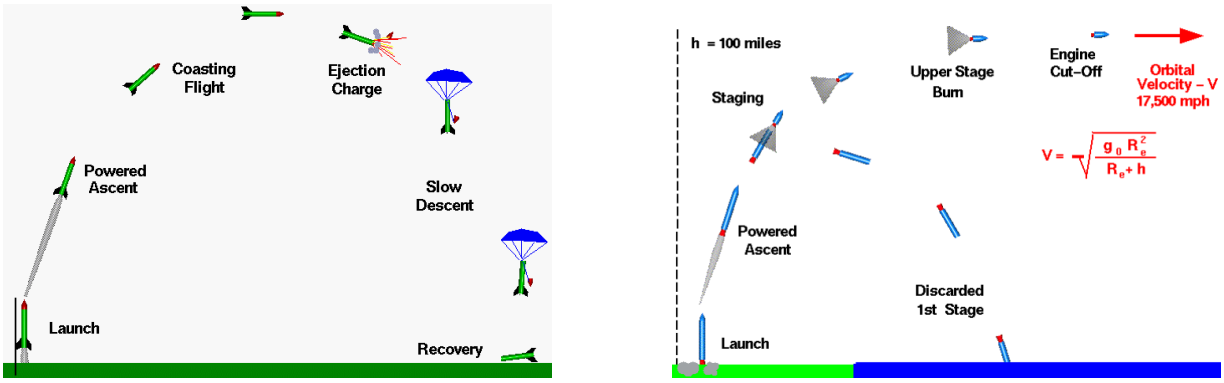


Figura 2.6: Lanzadores Más Pequeños a Comparación del Falcon 9 [1]

2.5. Ambiente de Vuelo

Los vehículos aeroespaciales, como los cohetes, tienen la capacidad de llevar a cabo dos tipos de vuelos. El vuelo convencional de un cohete atmosférico, representado en la sección A de la figura 2.7, comprende cuatro fases: lanzamiento, ascenso, punto de apogeo y descenso. Durante el vuelo, un modelo de cohete se encuentra sometido a cuatro fuerzas principales: peso, empuje, y las fuerzas aerodinámicas de sustentación y resistencia. La magnitud relativa y la dirección de estas fuerzas son determinantes en la trayectoria de vuelo del cohete.



(a) Vuelo Modelo (b) Vuelo a Orbita

Figura 2.7: Vuelo Cohete [11]

Sin embargo, en la sección B, se presenta la fase orbital del vuelo, donde la diferencia principal radica en que durante el lanzamiento, ya sea de una sola etapa o de múltiples etapas, la altura del apogeo se alcanza con la velocidad orbital, la cual está determinada por parámetros orbitales específicos. En esta sección, la primera etapa descartada continúa su vuelo en una trayectoria balística de regreso hacia la Tierra. En contraste, la etapa superior, de menor peso, sigue acelerando impulsada por su motor y se inclina gradualmente hacia la horizontal. A una altitud y velocidad cuidadosamente calculadas, el motor de la etapa superior se apaga, y tanto la etapa como la carga útil ingresan en órbita [12].

En ingeniería aeroespacial, es fundamental comprender cómo calcular la velocidad necesaria para mantener una órbita estable alrededor de la Tierra. Esta velocidad, denotada como V, depende de factores como la aceleración gravitatoria en la superficie terrestre (g_0), el radio de la Tierra (R_e), y la altitud sobre la superficie (h). La ecuación que describe este cálculo es la siguiente:

$$V = \sqrt{g_0 \cdot \frac{R_e^2}{(R_e + h)}} \tag{2.24}$$

Esta fórmula es esencial en el diseño de misiones espaciales y satélites, ya que

permite determinar la velocidad necesaria para mantener una órbita específica a una determinada altitud sobre la Tierra.

2.6. Sistema de Control

La imperativa necesidad de implementar sistemas de control en el ámbito aeroespacial, particularmente en el caso de cohetes no tripulados, reviste una importancia fundamental para la automatización completa del proceso, desde el instante mismo del despegue. A lo largo del vuelo del cohete, este se ve continuamente afectado por diversas perturbaciones, tales como el viento, desalineaciones en las fuerzas de empuje y asimetrías en la carga, entre otros factores ([15]. En virtud de estas consideraciones, el sistema de control debe ejercer una vigilancia constante sobre la trayectoria del vehículo y responder con celeridad a las variaciones detectadas por los sensores. Posteriormente, se requiere que corrija dichas desviaciones mediante el accionamiento de actuadores, con el fin de garantizar que el satélite alcance exitosamente la órbita planificada. Es importante resaltar que incluso la más mínima desviación podría conllevar a un significativo desplazamiento a lo largo de distancias astronómicas.

Dentro del ámbito de los sistemas de control, se reconocen dos enfoques fundamentales: el lazo abierto y el lazo cerrado. En este proyecto de investigación, nos hemos centrado en el de lazo cerrado. A lo largo del desarrollo de este estudio, se ha implementado un control de tipo clásico, el cual, como es universalmente aceptado, exige una estrecha correlación con un modelo matemático rigurosamente definido (referido como la planta o sistema), tal y como se esquematiza en la Figura 2.8. Adicionalmente a este enfoque, hemos emprendido una exploración del control basado en Lógica Difusa (Fuzzy Logic). Dado que este último enfoque constituye un sistema experto, nos habilita para llevar a cabo un control de naturaleza más ïntuitiva", que se basa en experiencias y no está estrictamente subordinado a un modelo matemático preciso, sino que emula el razonamiento de un experto humano.

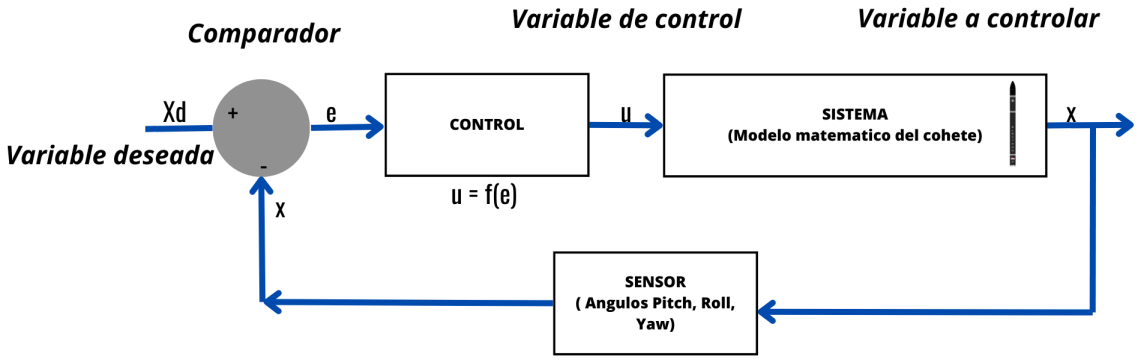


Figura 2.8: Sistema de Control Lazo Cerrado/Controlador Clásico y Fuzzy Logic

2.6.1. Sistema de Control PID

La implementación del bloque de control Proporcional Integral Derivativo (PID) se ejecutó de manera elemental, una vez se obtuvo el conocimiento fundamental relacionado con la planta que debía ser controlada mediante el sistema de actuación. A continuación, se presenta una ilustración del sistema básico de un controlador PID en la Figura 2.9.

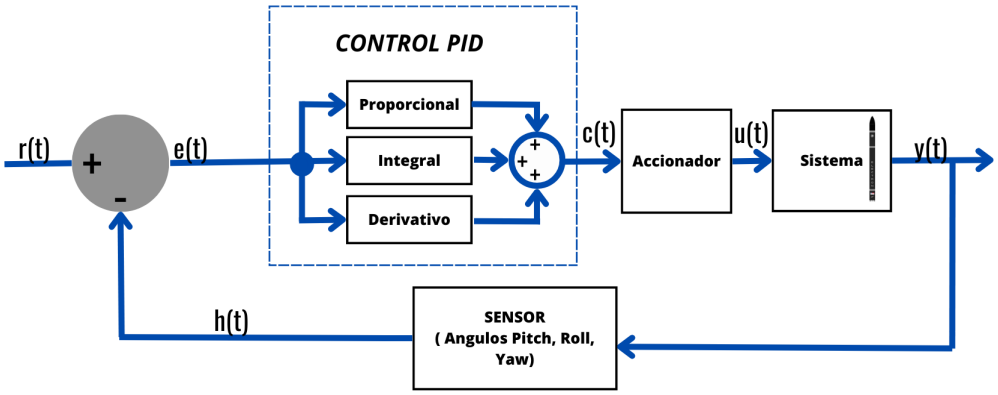


Figura 2.9: Controlador Proporcional Integral Derivativo

El principio fundamental que subyace al esquema de control Proporcional Integral Derivativo (PID) radica en su capacidad para influir en la variable manipulada mediante la combinación de tres acciones de control distintas: en primer lugar, la acción proporcional, donde la respuesta del control es directamente proporcional al error presente, definido como la diferencia entre la entrada deseada y la señal de retroalimentación; en segundo lugar, la acción integral, donde la respuesta del control es proporcional a la integral del error acumulado a lo largo del tiempo; y, por último, la acción derivativa, donde la respuesta del control es proporcional a la derivada del error respecto al tiempo[13].

En el control de sistemas dinámicos, especialmente en sistemas de control PID (Proporcional, Integral, Derivativo), la señal de control $u(t)$ se define mediante la siguiente ecuación:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{2.25}$$

donde:

- $u(t)$ representa la señal de control que afecta el sistema.
- $e(t)$ es el error entre la salida deseada y la salida real del sistema en el tiempo t.
- K_p, K_i, y K_d son los coeficientes de ganancia para los términos proporcional, integral y derivativo, respectivamente.

- La integral $\int_0^t e(\tau)d\tau$ representa la acumulación del error a lo largo del tiempo.
- La derivada $\frac{de(t)}{dt}$ mide la tasa de cambio del error.

Esta ecuación es fundamental en el diseño de controladores PID y desempeña un papel crucial en la regulación y estabilización de sistemas dinámicos en una amplia gama de aplicaciones industriales y de ingeniería.

Adicionalmente, la función de transferencia $G(s)$ es un componente esencial en el análisis y diseño de sistemas de control. Está definida por la ecuación:

$$G(s) = K_P \left(1 + \frac{1}{T_i s} + T_d s\right) \tag{2.26}$$

donde:

- K_P es la ganancia proporcional, que determina la influencia de la señal de error en la salida del sistema.
- T_i es la constante de tiempo integral, que controla la capacidad del sistema para corregir errores en estado estacionario.
- T_d es la constante de tiempo derivativa, que afecta la habilidad del sistema para anticipar cambios futuros en el error.

Esta ecuación captura la relación entre la entrada y la salida de un sistema dinámico en el dominio de Laplace, lo que permite predecir y ajustar su comportamiento en respuesta a diferentes señales de entrada.

El valor de $G(s)$ es fundamental para determinar la estabilidad, respuesta dinámica y precisión de un sistema de control. Por lo tanto, comprender y sintonizar los parámetros K_P, T_i, y T_d es esencial para diseñar un controlador efectivo y alcanzar los objetivos deseados en términos de rendimiento y estabilidad.

2.6.2. Fuzzy Logic (Controlador de Lógica Difusa)

En el controlador presente, se utiliza un formalismo matemático que tiene como finalidad representar el razonamiento característico de un experto humano. Por lo tanto, su aplicación se lleva a cabo en sistemas expertos y en aplicaciones vinculadas con la inteligencia artificial [7].

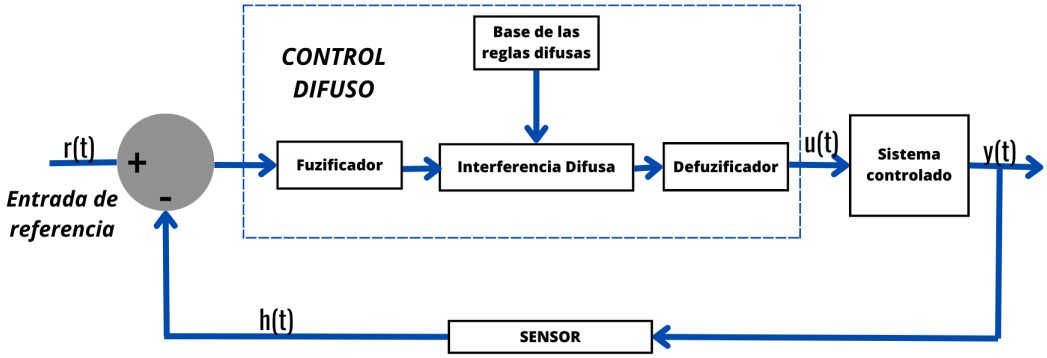


Figura 2.10: Controlador de Lógica Difusa Genérico

Este algoritmo permite la generación de una salida de control a partir de variables de entrada, que son procesadas mediante su ley de control. Al tratarse de un controlador de lazo cerrado, como algunos controladores convencionales, tiene la capacidad de mejorar los resultados a través de la derivación del error. Los sistemas de control difuso constan de cuatro componentes fundamentales: fuzzificación, inferencia, reglas difusas y defuzzificación [16].

2.7. Sensores

Los sensores desempeñan un papel esencial al proporcionar datos en entornos tanto físicos como virtuales, lo que, a su vez, posibilita la toma de decisiones mediante la actuación de dispositivos específicos. Entre estos sensores, se encuentran el acelerómetro y el giroscopio, que tienen la capacidad de medir los ángulos en los ejes Roll, Pitch y Yaw (X, Y, Z). Estas mediciones adquieren una importancia fundamental en el contexto de vuelos futuros, ya que permiten la concepción de cargas útiles que pueden funcionar con eficacia en las distintas direcciones de vuelo. Dado que los cohetes, en contraste con vehículos terrestres o marítimos, poseen la capacidad de moverse en tres dimensiones, la precisión en estas mediciones se torna crítica.

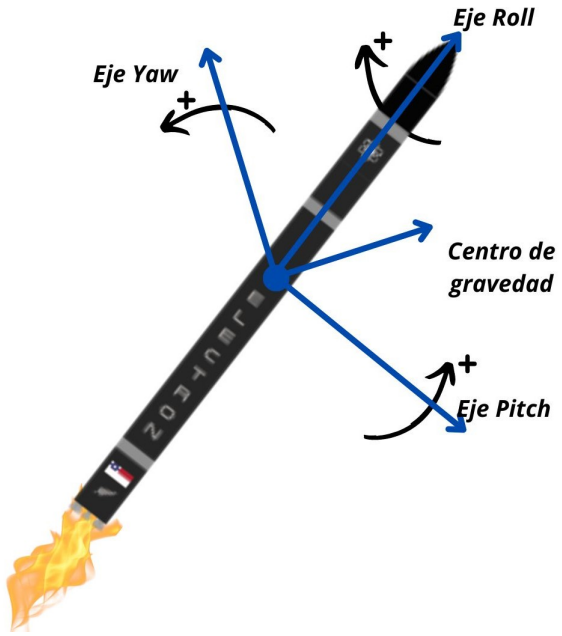


Figura 2.11: Ejes de Rotación de un Cohete

2.7.1. Sensor MPU6050

El sensor MPU6050 está equipado con una Unidad de Medición Inercial (IMU), un término genérico empleado para describir un dispositivo capaz de medir la velocidad, orientación y aceleración de un sistema determinado. Este sensor dispone de nueve ejes de movimiento, comprendiendo tres ejes con giroscopio y tres ejes con acelerómetro, junto con un procesador de movimiento digital extensible. Su interfaz es adaptable para conectarse con otros sensores a través de DMP, IIC o SPI, y su salida se traduce en una señal que representa los nueve ejes ([6]).

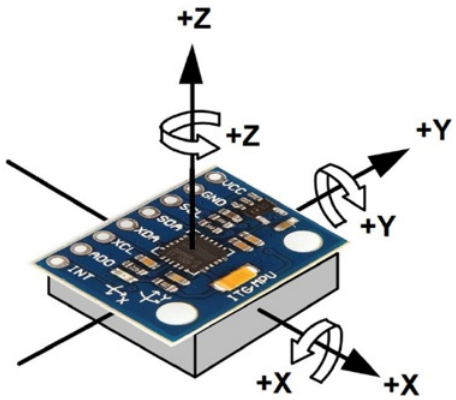


Figura 2.12: Ejes del Sensor MPU6050 [6]

La ventaja inherente de este sensor radica no solo en su amplia disponibilidad en la

industria electrónica, sino también en su capacidad para proporcionar los ángulos de inclinación al determinar su posición y orientación. Este sensor se encuentra ampliamente empleado en aplicaciones de detección de inclinación en sistemas de control de vuelo. No obstante, es imperativo recalcar que el sensor debe ser sometido a un proceso de calibración mediante métodos adecuados. Tras la calibración, para lograr resultados óptimos, se sugiere la implementación de un filtro, ya sea un filtro complementario debido a su sencillez en la aplicación o un filtro de Kalman. Es esencial tener en cuenta que el filtro complementario representa una simplificación del filtro de Kalman y omite por completo el análisis estadístico.

Este filtro complementario viene dado por la siguiente ecuación:

$$\theta = A \cdot (\theta_{prev} + \theta_{gyro}) + B \cdot \theta_{accel}) \tag{2.27}$$

En el contexto en el que A y B son dos constantes debidamente calibradas, la calibración del filtro puede llevarse a cabo de manera directa al variar los valores de A y B, siempre y cuando se garantice que la suma de dichos valores sea igual a 1.

Capítulo 3

Metodología de Implementación y Resultados

Basándonos en los estudios previos, se definieron cuatro actividades principales que se abordaron de manera modular. Estas actividades se diseñaron e implementaron de forma secuencial, a medida que se completaban las etapas precedentes. Además, se presentarán los resultados obtenidos a través de las simulaciones de cada uno de los sistemas empleados para alcanzar los objetivos establecidos en este proyecto de investigación.

3.1. Diagrama de Cuerpo Libre

Iniciamos un análisis de cuerpo libre en un cohete genérico con el objetivo de identificar las fuerzas que actúan sobre él y obtener información esencial acerca de su estabilidad durante el inicio de su trayecto. Nuestro objetivo es determinar si el cohete experimentará una mayor estabilidad o, por el contrario, si tendrá un vuelo inestable.

- **Centro de Presión (CP):** El centro de presión se define como el punto de convergencia de todas las fuerzas aerodinámicas normales que afectan a un modelo de cohete durante su vuelo. De manera más sencilla, podemos describirlo como el punto en el que se concentra la fuerza resultante de la presión ejercida por el aire sobre el cuerpo del cohete.
- **Centro de Gravedad (CG):** El centro de gravedad, también referido como centro de masas, es el punto donde se encuentra concentrada la totalidad del peso del cohete. A medida que el propelente del motor se va consumiendo durante el vuelo, este punto experimenta variaciones.

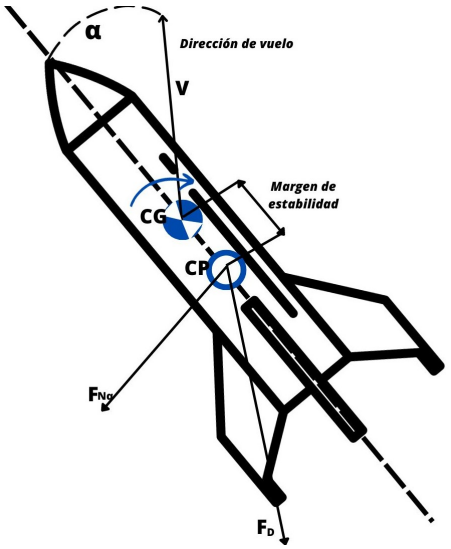


Figura 3.1: Diagrama de Cuerpo Libre

- **Estabilidad:** La estabilidad se establece en función de la separación entre el centro de presión y el centro de gravedad. Por lo general, la distancia mínima se encuentra determinada por el diámetro del cuerpo del cohete
- **Dirección de Vuelo:** La dirección de vuelo se define mediante el ángulo de ataque, representado como α, y la orientación de vuelo, que se describe en relación al vector de velocidad V con respecto al centro de gravedad.
- **Fuerza de Arrastre:** La fuerza de arrastre se refiere a la componente aerodinámica que actúa directamente sobre el centro de presiones y se opone en la dirección del vuelo del cohete.
- **Fuerza Normal:** Esta componente de fuerza incide de manera directa sobre el centro de presiones, dando lugar a una rotación alrededor del centro de gravedad del cohete y ocasionando un momento de torsión durante el vuelo. La magnitud de la fuerza normal aumenta de manera proporcional al incremento del ángulo de ataque.

3.2. Desarrollo del Modelo Matemático y Sistema de Control Adecuado

Una vez adquiridos los conocimientos fundamentales en la mecánica, tanto en su vertiente dinámica como estática, del vuelo de cohetes, se seleccionó el Electron Rocket, desarrollado por la empresa Rocket Lab, como punto de referencia. Este cohete, con un diseño de múltiples etapas, específicamente de dos, se destaca por su capacidad para transportar cargas útiles de hasta aproximadamente 300 kilogramos a la órbita

terrestre baja (LEO). Una característica sobresaliente de este vehículo espacial reside en su enfoque innovador en la fabricación de motores de cohetes mediante tecnología de impresión 3D, un avance significativo en la industria aeroespacial.

A pesar de que su primera etapa no es completamente reutilizable, Rocket Lab está inmersa en un programa de recuperación con el propósito de reacondicionar y reutilizar estas etapas en futuras misiones de lanzamiento[19]. Con una altura total de aproximadamente 17 metros, equivalente a 56 pies, el Electron Rocket se presenta como una opción altamente efectiva para el lanzamiento de cargas útiles de menor envergadura hacia el espacio. Este enfoque abre nuevas perspectivas en la industria de los satélites y la exploración espacial en una escala más reducida en comparación con lo que se había visto en el mercado hasta ese momento.

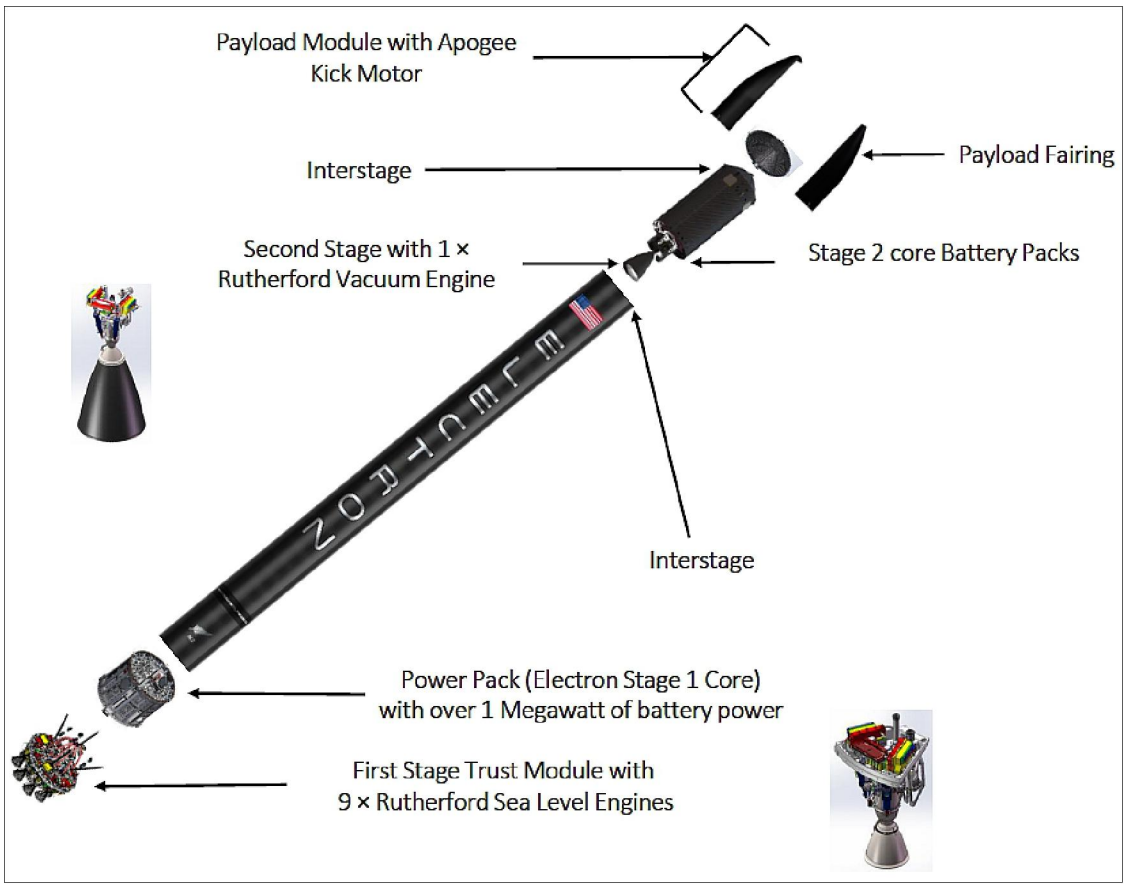


Figura 3.2: Electron Rocket[19]

Por las razones expuestas y con el objetivo de conferir mayor realismo al sistema, al mismo tiempo que se mantiene su simplicidad, se decidieron utilizar datos auténticos del Electron Rocket proporcionados por el fabricante. Estos datos se emplearon en la ejecución y simulación del modelo matemático a través del programa MATLAB R2023a. Los parámetros pertinentes se detallan en la Tabla 3.1.

Descripción	Parámetro	Valor
Coeficiente de fricción dinámico	C_d	0,25
Matriz de inercia	I	eye(3)
Masa de la primera etapa	m_{s1}	9250+950 kg
Masa de la segunda etapa	m_{s2}	2050+250 kg
Masa propelente	m_p	150 kg
Caudal masico de escape	$\dot{m}_{se}$	8 kg/s
Velocidad de descarga	v_e	3050 m/s
Presión de escape	p_e	100000 Pa
Motor termico primera etapa	$m_{e_{s1}}$	9
Motor termico Segunda etapa	$m_{e_{s2}}$	1

Tabla 3.1: Parámetros Considerados

Para llevar a cabo esta misión, se utilizó el «Toolbox Aerospace Blockset» de MATLAB, una herramienta que proporciona arquitecturas de modelos estándar para la creación de plataformas de vehículos reutilizables. Estos modelos facilitan análisis de vuelo y misiones, estudios conceptuales, diseño detallado de misiones, desarrollo de algoritmos de orientación, navegación y control (GNC), pruebas de integración de software, así como pruebas de hardware en bucle cerrado (HIL) aplicables a vuelo autónomo, radar y comunicaciones [9].

Los bloques principales sometidos a prueba en este Toolbox se detallan en la Tabla 3.2 y 3.3.

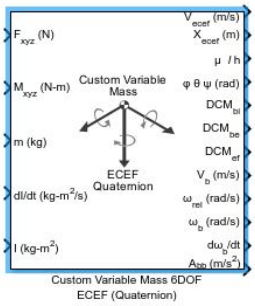
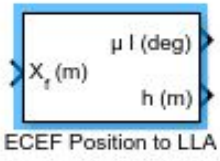
Bloques	Características
	Implementa una representación de cuaternión de ecuaciones de movimiento de seis grados de libertad de masa variable personalizada en coordenadas fijas con centro en la Tierra (ECEF).
	Convierte un vector de posición ECEF de 3 por 1,en latitud geodésica,longitud y altitud encima del elipsoide planetario.

Tabla 3.2: Bloques de Simulink Toolbox Aerospace Blockset Sección 1.

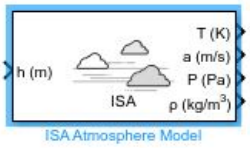
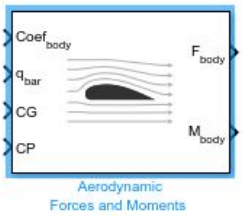
Bloques	Características
	Implementa la representación matemática de los valores atmosféricos estándar internacionales para la temperatura ambiente, presión, densidad y velocidad del sonido para la altitud geopotencial de entrada.
	Calcula las fuerzas y momentos aerodinámicos alrededor del centro de gravedad.

Tabla 3.3: Bloques de Simulink Toolbox Aerospace Blockset Sección 2.

3.3. Configuración de Microcontrolador y Parámetros de Interfaz

Se definió la configuración operativa del microcontrolador Arduino Mega 2560 en el contexto de las pruebas del sistema de control para un cohete a escala. Se procedió a verificar minuciosamente que la configuración estuviera correctamente establecida, incluyendo parámetros relativos a las interfaces, tales como puertos de entrada y salida, velocidad de comunicación, protocolos de comunicación, entre otros. Esto se llevó a cabo con el objetivo de garantizar que el microcontrolador operara de manera eficiente dentro del sistema.

Figura 3.3: Entorno de Desarrollo[9]

3.3.1. Conexiones Esenciales: Interfaces de Comunicación con el Microcontrolador

La comunicación entre Arduino y MATLAB Simulink implicó la creación de un modelo en Simulink que hacía uso de bloques diseñados específicamente para esta-

blecer interacciones con Arduino. Esto se pudo lograr a través de la implementación de una comunicación serial personalizada o aprovechando el soporte de hardware integrado en Simulink.

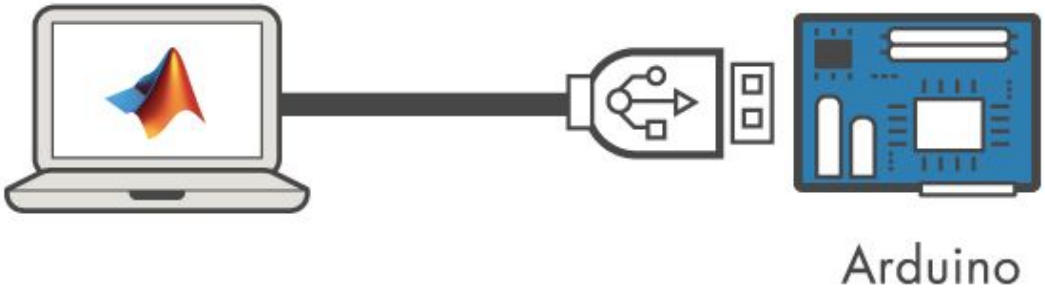


Figura 3.4: Paquete de Soporte MATLAB para Arduino[9]

El paquete de soporte de MATLAB para Arduino habilita la creación de programas en MATLAB que efectúan lecturas y escrituras de datos en el Arduino, al mismo tiempo que acceden a dispositivos periféricos como motores, LED y dispositivos I2C. Gracias a la naturaleza interpretada y de alto nivel de MATLAB, se simplifica sustancialmente el proceso de prototipado y refinamiento de algoritmos en el proyecto Arduino. Además, los resultados de las operaciones de entrada/salida se pueden observar de manera inmediata. MATLAB dispone de una extensa biblioteca de funciones integradas en matemáticas, ingeniería y visualización que están a su disposición para la programación de Arduino [10].

A continuación, se enumeran los paquetes de soporte requeridos para habilitar la comunicación en tiempo real entre Arduino y MATLAB.:

Biblioteca	**Características**
Simulink Support Package for Arduino Hardware	Tecnología de diseño basada en modelos para crear sistemas integrados en Arduino, desde la simulación hasta la implementación.
MATLAB Support Package for Arduino Hardware	Funciones matemáticas, de ingeniería y de trazado integradas que se incluyen con MATLAB para analizar y visualizar los datos recopilados de Arduino.

Tabla 3.4: Paquetes de Soporte de Simulink

Con el propósito de garantizar una comunicación efectiva entre MATLAB y Arduino, se incorporaron los siguientes bloques en nuestro sistema, lo que facilitó la interacción con el microcontrolador de manera significativa.

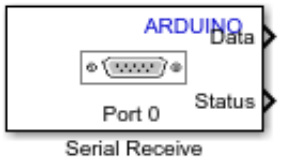
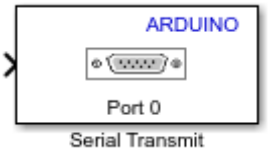
Bloques	Características
	Recibe una matriz de datos a través del puerto serie especificado y la envía a la salida del bloque de datos.
	Transmite los datos almacenados en el búfer hacia el puerto serie especificado.

Tabla 3.5: Bloques de Simulink

Configurando el bloque de recepción serial:

- **Encabezado:** El encabezado permite a Simulink identificar el inicio del mensaje. Aunque no es estrictamente necesario para la comunicación, se recomienda ampliamente su uso, ya que ayuda a evitar posibles problemas de sincronización. En este ejemplo, se utilizó el byte«0», pero se puede elegir cualquier otro.
- **Terminator:** El terminator, al igual que el encabezado, marca el final del paquete de datos.
- **Tamaño de Datos:** Si está enviando solo un valor float desde Arduino, se utiliza [1 1]. Sin embargo, se puede ajustar a [1 2] o [1 N], donde N representa el número de valores float que recibe de la comunicación serie.
- **Tipo de Datos:** Si se está transmitiendo valores float desde Arduino, se debe asegurar de seleccionar el tipo de datos 'single' en Simulink, ya que ambos tipos de variables son intrínsecamente equivalentes (números de punto flotante con una longitud de 4 bytes).

Se debe recordar que el tipo de datos y el tamaño de los datos están relacionados; por lo tanto, si se configura el tamaño de los datos en [1 3], Simulink esperará recibir 3 * 4 bytes = 12 bytes en cada paso.

Configurando el bloque de Trasmisión serial:

- **Paso:** Generar la señal que se desea enviar.
- **Retención de orden cero:** Establecer la velocidad de envío de la simulación.
- **Single** Convertir la señal a single(4 bytes),tal como se recibe del microcontrolador double (ambos tipos son equivalentes).
- **Byte Pack** Establecer en uint32 ya que 1 single es 4 bytes, es decir, 1 uint32.
- **Serial Send** Envía los bytes. Se puede agregar un encabezado y un terminador si se desea, pero hay ningún problema en enviar estos datos sin ellos.

3.3.2. Asociar los parámetros acordes con la interfaz y el microcontrolador

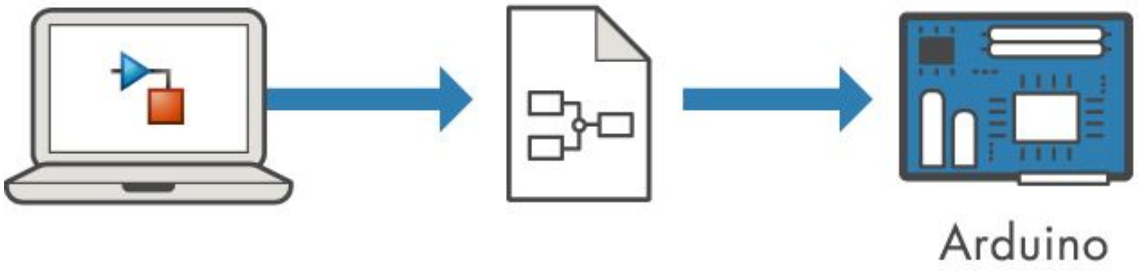


Figura 3.5: Generación Automática de Código[9]

El Arduino utiliza el protocolo de transferencia de datos UART, el cual opera a niveles lógicos TTL de 0V o 5V. Dado que los puertos USB funcionan con niveles lógicos distintos, se hace necesario emplear un convertidor serie USB a TTL basado en un circuito integrado (IC) para permitir la conectividad de Arduino con la computadora.

Figura 3.6: Adaptador USB a TTL

Para establecer la conexión entre el IC convertidor serie USB a TTL y Arduino, se deben seguir los siguientes pasos:

1. Enchufar el conector USB macho al puerto USB del PC.
2. Conectar los cables TX, RX, DTR, GND y 5V a los pines correspondientes de Arduino.
3. Asegurarse de que la velocidad en baudios seleccionada sea compatible con el microcontrolador.
4. Si es necesario, descargar e instalar los controladores IC PL2302 en el PC.

Estos pasos garantizan una conexión adecuada entre el convertidor USB a TTL y Arduino, permitiendo una comunicación efectiva entre ambos dispositivos.

3.4. Diseño e Implementación del Sistema Hardware In The Loop Simulation

Se ha implementado un sistema de Hardware en el Bucle (HIL) mediante un simulador en tiempo real basado en un microcontrolador Arduino. Este sistema posee aplicaciones relevantes tanto en el ámbito industrial como en el educativo. Específicamente, en contextos que hacen uso extendido de sistemas de accionamiento eléctrico, los cuales requieren niveles superiores de rendimiento, fiabilidad y capacidad de variación de velocidad para optimizar la controlabilidad.

En este sentido, incluso en la industria aeroespacial, donde los cohetes encuentran un amplio uso debido a sus características de carga y facilidad de adquisición de datos, el control del hardware se lleva a cabo a través del entorno MATLAB/Simulink. Este entorno ejecuta los circuitos del hardware en intervalos de tiempo real. El microcontrolador Arduino se programa mediante MATLAB/Simulink para generar ángulos a través de quaterniones, los cuales son transmitidos a la placa para que el controlador realice ajustes automáticos. Una vez completado este proceso, los datos se reintegran al modelo matemático alojado en MATLAB/Simulink, haciendo uso de la metodología HIL en tiempo real.

A continuación, se enumeran los elementos necesarios para establecer la comunicación Hardware-in-the-Loop (HIL) entre el microcontrolador y el software. Para garantizar una comunicación efectiva, resulta imperativo que el Arduino esté debidamente conectado a la computadora y que se seleccione el puerto de comunicación apropiado (COM9 para Arduino y COM12 para el TTL).

Una vez realizado este paso, es necesario seleccionar la velocidad en baudios para la comunicación con el Arduino. Algunos valores estándar comunes son **9600** y **115200** baudios. Por lo general, esta configuración se establece en la función setup() del código Arduino.

Configuración UART

- **Nombre del puerto COM:** especificar el nombre del puerto COM asociado con el dispositivo (igual que anteriormente)
- **Velocidad de transmisión:** establecida en 115200. Asegurarse de tener la misma velocidad en baudios programada en el microcontrolador.

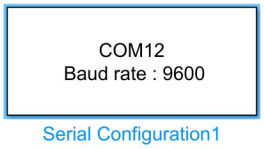
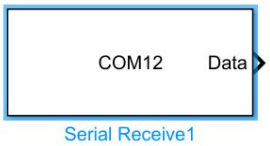
Bloques	Características
	Configura los parámetros para un puerto serie que puede utilizar para enviar y recibir datos.
	El bloque de recepción serie configura y abre una interfaz al puerto serie especificado.
	El bloque de envío serie configura y abre una interfaz al puerto serie especificado.
	El bloque Byte Pack convierte una o más señales de tipos de datos seleccionables por el usuario en una sola.

Tabla 3.6: Bloques de Simulink para la Comunicación HIL

- **Bits de datos:** configurados en 8bits.
- **Paridad:** ninguna.
- **Bits de parada:** configurado en 1bit de parada.
- **Orden de bytes:** little endian.
- **Control de flujo:** ninguno.
- **Tiempo de espera:** 10

3.5. Visualización del Comportamiento de Vuelo de un Cohete

Dentro del marco de este proyecto, se implementó un modelo de simulación con el propósito de analizar el comportamiento de vuelo de un cohete. Se utilizaron tanto simulaciones convencionales como pruebas de hardware en tiempo real. El modelo se basó en formulaciones matemáticas que consideraban la influencia de la propulsión, la interacción con la atmósfera y el efecto de la gravedad en la trayectoria, así como otros factores pertinentes.

En este contexto, partiendo del modelo matemático general del cohete y considerando el comportamiento físico junto con la incorporación de los parámetros específicos del Electrón Rocket, se procedió a determinar la mecánica de vuelo utilizando el simulador en Matlab-Simulink. Se elaboró un diagrama de bloques, como se ilustra en la figura 3.7, para llevar a cabo la modelización del cohete de múltiples etapas. La primera etapa permanece inactiva hasta que el propelente del motor se haya agotado por completo a través de sus 9 propulsores. Solo entonces se permite la activación de la segunda etapa.

Sin embargo, al explorar las herramientas proporcionadas por la empresa MathWorks, se tuvieron en cuenta las condiciones ambientales que pueden influir en la trayectoria o dirección del cohete, como ráfagas de viento cortante. Estos fenómenos pueden ocasionar desviaciones en el rumbo del cohete, que se expresan en grados. Por ejemplo, un cohete con aletas estáticas puede tolerar cierto grado de desviación antes de que las aletas logren estabilizarlo. No obstante, existen situaciones en las que las desviaciones pueden ser más pronunciadas, superando los 15 grados en relación al eje de la trayectoria. Si esto ocurre, el cohete perderá el control, experimentará una pérdida de orientación y, como consecuencia, se precipitará, dando lugar a un fracaso total de la misión. Esto podría ocasionar retrasos en la investigación u objetivos de lanzamiento, además de significativas pérdidas económicas.

La comprensión y mejora del rendimiento de los cohetes en diferentes condiciones y fases de vuelo se ha convertido en una herramienta crucial en el proceso de diseño y desarrollo de sistemas espaciales. La metodología empleada ha permitido una comprensión detallada de este comportamiento. En el subsistema conocido como «Electron Rocket», se ha utilizado la técnica de Quaterniones para simular un escenario de dos etapas con masa variable.

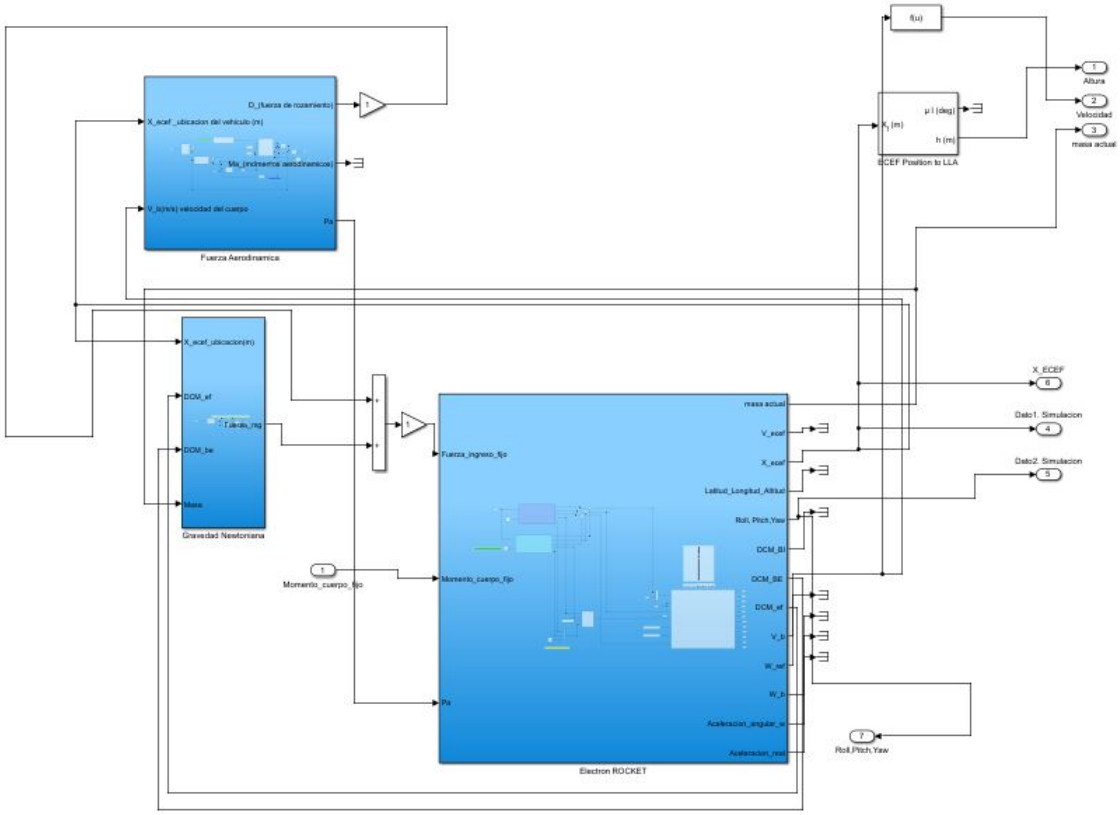


Figura 3.7: Diagrama del Modelo Integrando Factores Externos

Al concluir el proceso de modelado, han surgido consideraciones adicionales que incluyen el agotamiento del combustible, la velocidad de vuelo durante la trayectoria del cohete, la altura máxima alcanzada (apogeo) y la recopilación de datos de un sensor virtual para obtener los ángulos de inclinación del cohete, entre otros factores.

No obstante, resulta relevante señalar que este modelo requiere la influencia de un sistema de control apropiado para su finalidad o aplicación específica. En este contexto, con el fin de mantener la complejidad en su nivel más bajo, se ha optado por la implementación de un sistema de control PID. Dicho sistema, operando en modalidad de bucle cerrado con retroalimentación a través de los sensores virtuales previamente mencionados, posee la capacidad de estabilizar el sistema.

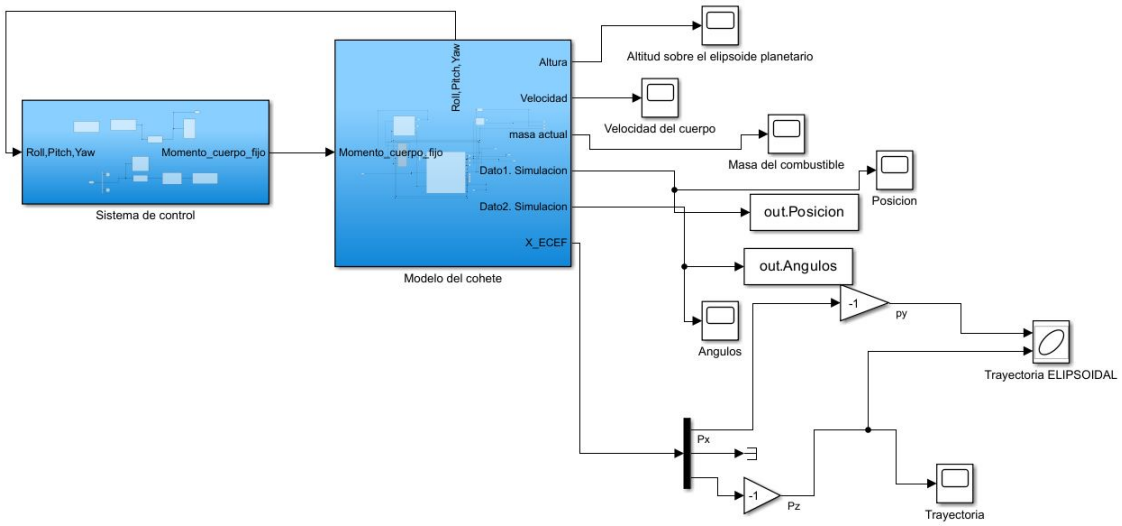


Figura 3.8: Esquemático del Controlador y el Modelo Matemático

El modelo en Simulink se divide primordialmente en dos subsistemas: el bloque denominado «Sistema de Control», responsable del intercambio de datos, y el componente denominado «Modelo del Cohete» . Este último implementa una abstracción simplificada de la variación de la masa de la aeronave debido a las fuerzas externas. Este subsistema recibe como entrada los ángulos de Euler controlados (provistos por el microcontrolador) y genera el momento resultante, lo que conduce a una actualización en tiempo real de la velocidad aérea. Posteriormente, los ángulos de Euler se transmiten nuevamente al bloque «Sistema de Control»para aplicar una ley de control en el hardware, como se ilustra en la figura 3.8.

Sin embargo, es fundamental aclarar que la función de transferencia específica para nuestro sistema de control PID se expresa de la siguiente manera:

$$G(s) = \frac{-16.68s^4 - 128s^4 - 5.267e05s^2 - 2.182e06s - 2.222e05}{s^4 + 148.4s^3 + 2.324e04s^2 + 1.068e06s + 8.45e06} \tag{3.1}$$

En esta expresión, tanto el numerador como el denominador son polinomios en "s", y la estabilidad del sistema se determina mediante la ubicación de las raíces del denominador, que representan los polos de la función de transferencia.

Es importante destacar que el ejecutor de estabilidad dio un resultado del 88.65 % en la función de transferencia, si embargo, no constituye una métrica convencional para evaluar la estabilidad de un sistema en el contexto de la teoría de control o sistemas dinámicos. La estabilidad del sistema se evalúa principalmente mediante la ubicación de los polos en el plano complejo. Cuando todos los polos se ubican en el semiplano izquierdo (parte real negativa) del plano complejo, el sistema se considera estable. En caso de que algunos polos se encuentren en el semiplano derecho (parte real positiva),

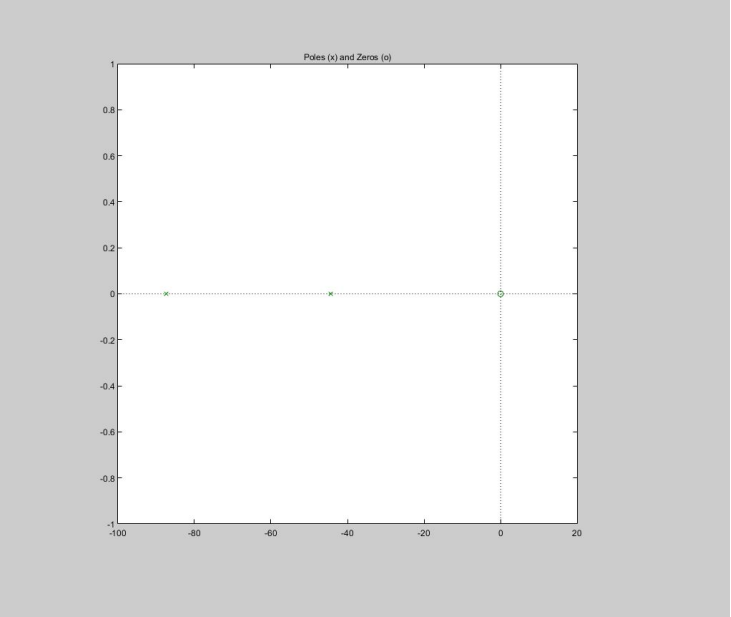


Figura 3.9: Polos y Ceros del sistema

el sistema puede ser inestable.La figura 3.9, muéstrala disposición de los 4 polos y 4 ceros del sistema, lo cual es un aspecto crítico para la evaluación de su estabilidad y comportamiento.

3.6. Conectando el Hardware y el Software: Iniciando una Simulación HIL para Sistemas Integrados

La primera etapa para implementar el sistema HILS y comprender su funcionamiento fue llevar a cabo el diseño de un HILS para una multiplicación. En la figura 3.10, podemos ver el esquemático del modelo en simulink para este sistema.

Cabe resaltar que al realizar el HILS, se sugiere trabajar en dos ambientes distintos de Simulink. Uno de ellos se destina para enviar datos y el otro para recibirlos en serie mediante el hardware Arduino.

En este esquema que vamos a implementar, los modelos:

- *arduino_serial_sendreceive*
- *arduino_serial_send_println*

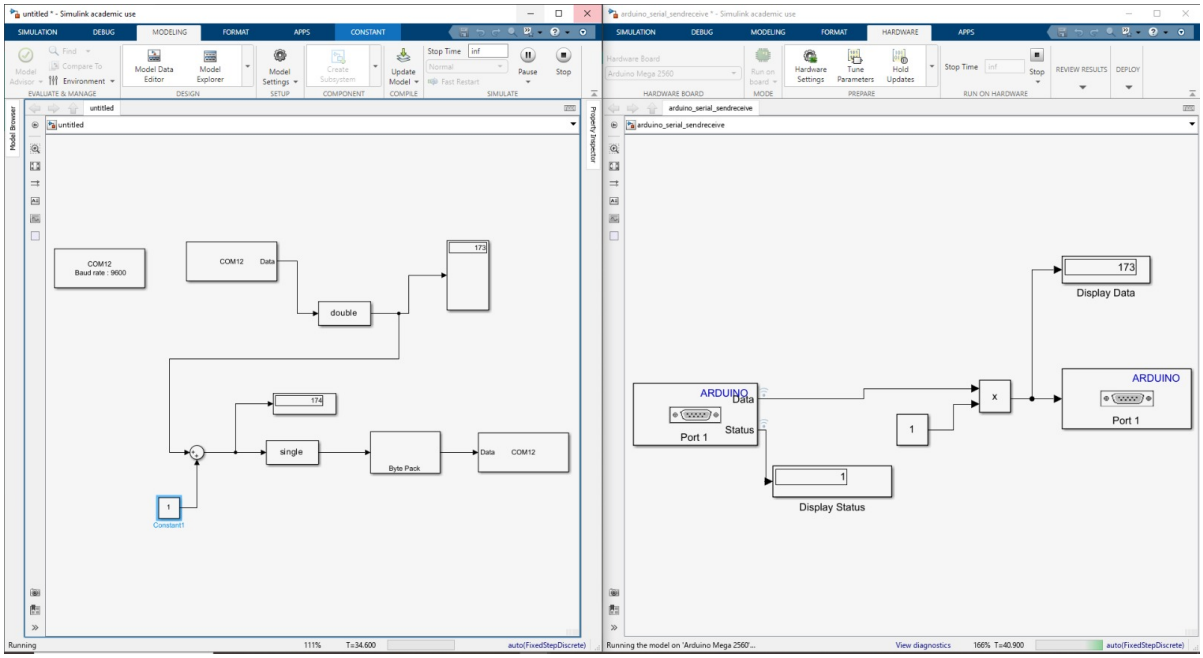


Figura 3.10: Esquemático Simulación HILS de Multiplicación

Para enviar y recibir datos en serie mediante Simulink. Estos modelos hacen uso de bloques de **Serial Transmit** y **Serial Receive** para intercambiar datos.

En el modelo arduino_serial_sendreceive, el pin TX1 transmite datos en serie al pin RX1 del hardware Arduino. Este modelo se encuentra configurado para operar en modo externo. Por otro lado, en el modelo arduino_serial_send_println, el hardware Arduino envía datos al PC a través del puerto serie 0 (conexión USB) del propio hardware Arduino. Este modelo se encuentra configurado para operar en modo Normal. En este modo, el modelo se implementa en el hardware Arduino.

Una vez implementado este modelo convencional observado en la figura 3.10, se procede a realizar la misma metodología para la implementación del HIL con el esquemático de la figura 3.8, dando como resultado el esquematico de la figura 3.11.

La disposición de cada uno de los bloques en la figura 3.11 corresponde exactamente a la configuración descrita en detalle en la sección correspondiente de la Tabla 3.6.

En el presente informe, se presentan los resultados obtenidos en forma de gráficos que destacan aspectos críticos del vuelo del cohete. Estos resultados, generados a partir del modelo, proporcionarán una visión detallada del desempeño de la nave espacial a lo largo de su trayectoria.

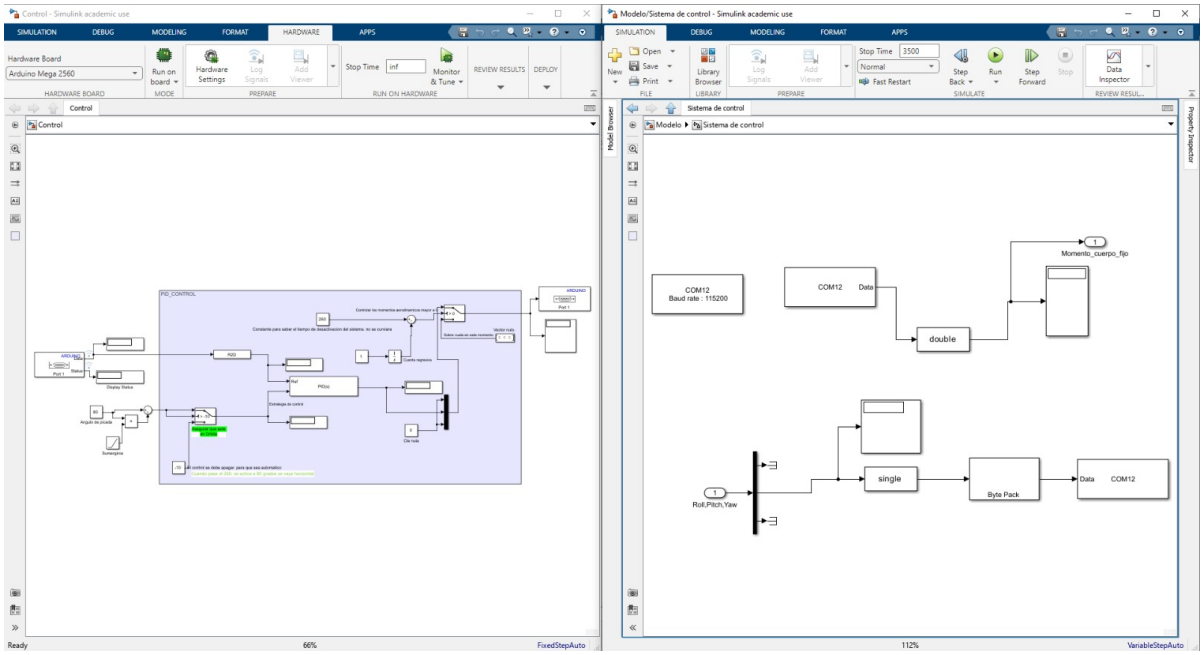


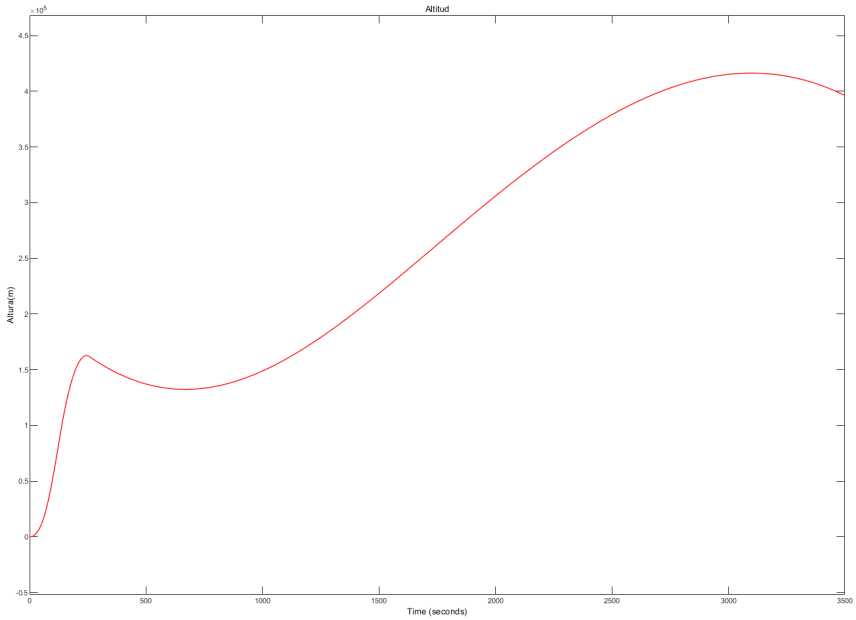
Figura 3.11: Esquemático Simulación HIL para el Modelo de Comportamiento

3.6.1. El Comportamiento Mediante Simulación Convencional y La Optimización del Hardware en Tiempo Real: Una Perspectiva Inicial

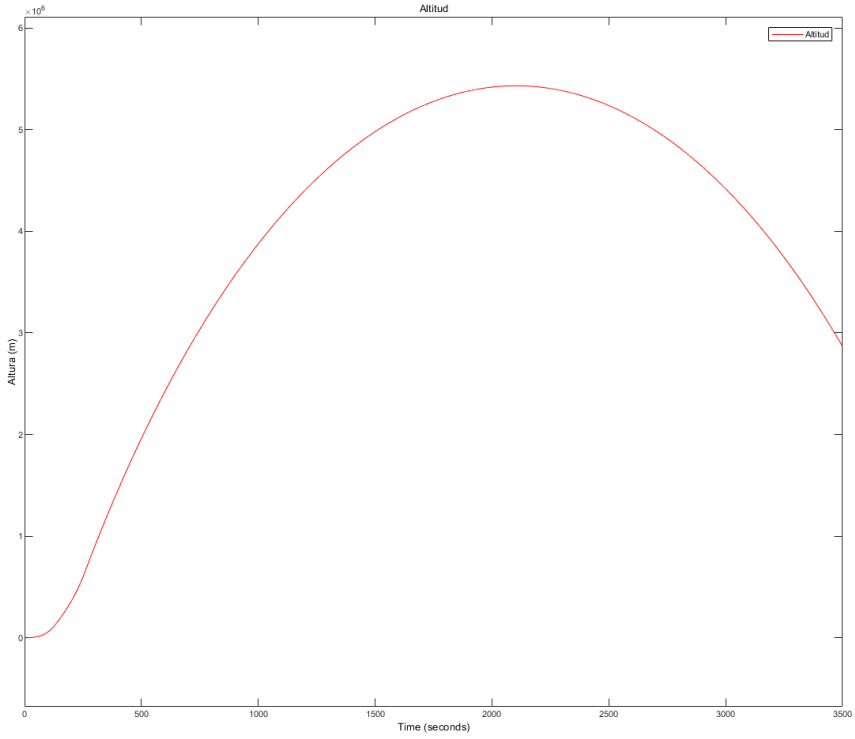
En el contexto de este proyecto, hemos dado un paso fundamental al emplear una simulación convencional y de Hardware-In-The-Loop (HIL) en tiempo real, para entender y analizar el comportamiento del cohete en su vuelo. Esta metodología nos brinda una valiosa perspectiva sobre el rendimiento de la nave en una variedad de condiciones y etapas durante su trayectoria.

En esta sección se presentan gráficas que ilustran de manera detallada la evolución de cinco parámetros críticos durante el vuelo del cohete: altitud (Fig 3.12), velocidad (Fig 3.13), masa (Fig 3.14), ángulos de Euler (Fig 3.15) y trayectoria elipsoidal (Fig 3.17). Estos gráficos proporcionan una representación visual esencial del rendimiento de la nave a medida que avanza en su vuelo. Cada uno de estos parámetros desempeña un papel crucial en el análisis y diseño de sistemas de cohetes, y su comportamiento dinámico es fundamental para evaluar el éxito de la misión. A través de estas representaciones visuales, podremos examinar de cerca cómo el cohete responde a las diferentes fases de su vuelo, desde el despegue hasta que llega a su punto de apogeo.

En las gráficas 3.12 se aprecia una marcada disparidad en el punto de apogeo, a pesar de que la trayectoria de vuelo del cohete sonda experimenta una variación de 10° grados. En la figura 3.12a, se registra un punto máximo de apogeo de **420 km** a



(a) Altitud Simulación Convencional



(b) Altitud Simulación Hardware en Tiempo Real

Figura 3.12: Dinámica de Vuelo: Altitud

los **3.250** segundos, mientras que en la figura 3.12b, el punto más alto se sitúa en **5.500 km** a los **2.000** segundos. Esto pone de manifiesto que, en tiempo real, el mismo modelo alcanzaría un apogeo superior,aproximadamente un **7.6 %** más elevado.

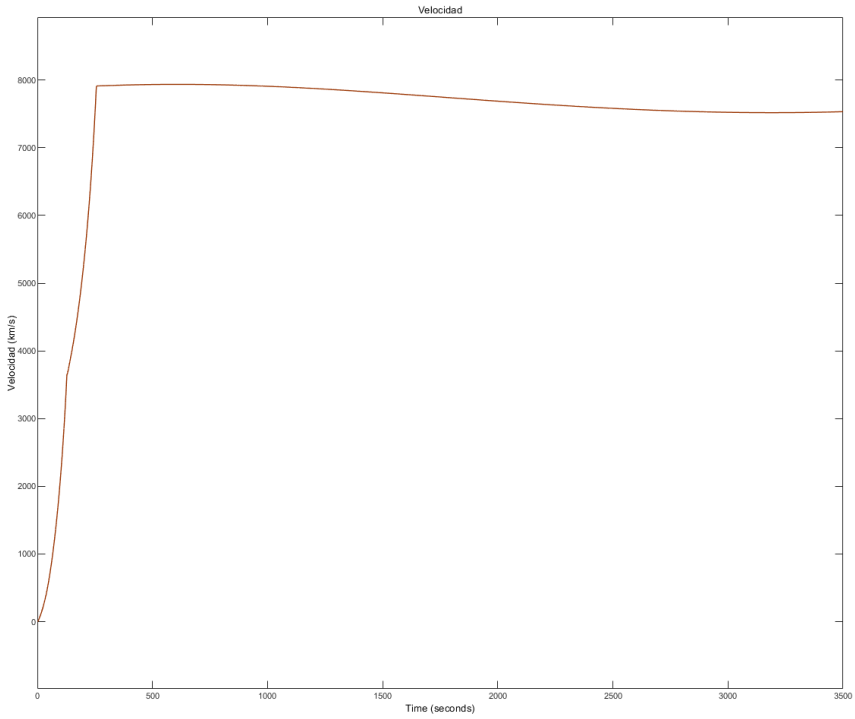
Asimismo, durante la simulación, en el momento de la ignición de la primera y segunda etapa, se evidencian de manera notable los cambios en la velocidad y altitud del cuerpo, como consecuencia de los ángulos de lanzamiento y la velocidad angular transmitida por el planeta. En este escenario, el cohete en la simulación convencional, hasta ese momento, se encuentra emergiendo de la región de la termosfera tras completar una órbita. En contraposición, el cohete en la simulación de hardware en tiempo real alcanza con precisión la exosfera, logrando una subórbita, la cual es la ubicación típica de los satélites para evitar las auroras boreales.

En la gráfica 3.13b se ilustra la variación de la velocidad total experimentada por el cohete a lo largo de toda su trayectoria de vuelo. Se resaltan dos puntos de importancia notable: el punto de máxima presión dinámica, que se alcanza al concluir la fase de combustión, y el apogeo, donde las fuerzas gravitatorias incitan al cohete a reiniciar su ascenso en velocidad hasta el punto en el que esta aumenta y da origen a un frenado aerodinámico. En otras palabras, a un mayor ángulo corresponde una mayor resistencia aerodinámica.

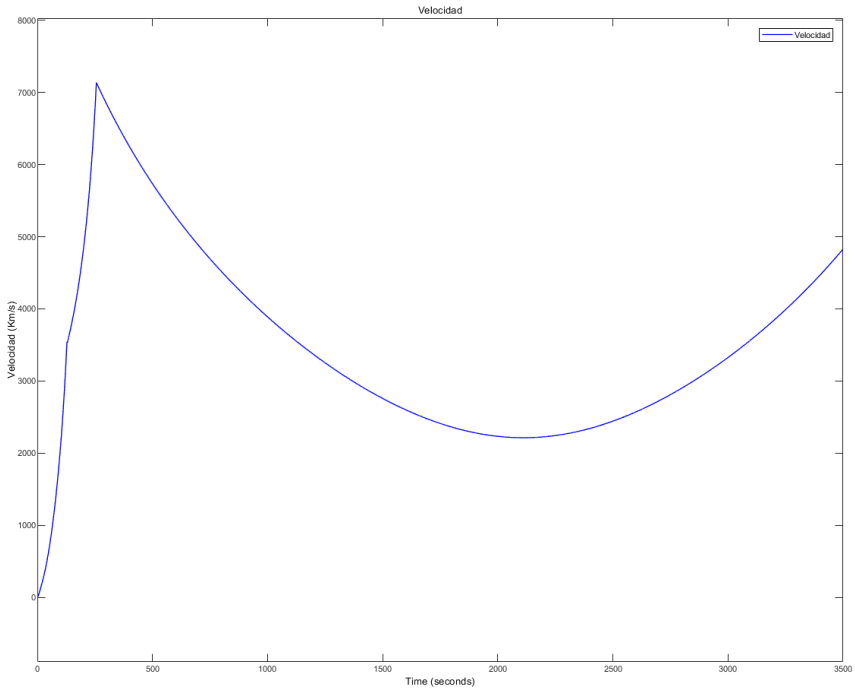
De manera similar, se aprecia en las figuras 3.13 que la agotación de la primera etapa tiene lugar a una velocidad de **3.5 km/s**. No obstante, al momento de la ignición de la segunda etapa, se observa que el punto máximo de velocidad en su apogeo es de aproximadamente **7.8 km/s** en la simulación convencional, mientras que en la simulación de Hardware en tiempo real es de **7.1 km/s**. Este fenómeno evidencia que en la primera gráfica, al no estar sujeta a una simulación en tiempo real, la respuesta del controlador es más pronunciada, permitiendo que su velocidad se estabilice en aproximadamente **7.5 km/s**. Como se mencionó previamente, esta simulación logra completar una órbita.

En contraposición, en el caso de la velocidad en el Hardware in the Loop (HIL), al estar sujeta al tiempo real y a una tasa de transmisión de datos considerablemente alta, una vez alcanza su punto de máxima apogeo, su velocidad comienza a descender de manera notable hasta **2.5 km/s**, coincidiendo en tiempo con el apogeo máximo de la altitud, tal como se evidencia en la figura 3.12b. Este descenso de velocidad provoca que la altitud en el HIL sea notablemente superior en comparación con la simulación convencional , debido a la disminución de su velocidad. De manera similar, a mayor altitud, menor velocidad, lo que indica que el vehículo está ingresando a una zona suborbital debido a las fuerzas gravitacionales que actúan sobre él, originando así su posicionamiento.

Ahora bien, en el momento del despegue, el cohete activó sus nueve motores de la primera etapa, propulsados por combustible líquido, manteniéndolos operativos durante aproximadamente **120** segundos con una masa de carga útil de **950 kg** más la masa del combustible de **9250 kg**. Alrededor de cinco segundos antes de que los nueve propulsores alcanzaran su agotamiento, se procedió a la separación de la estructura



(a) Velocidad Simulación Convencional



(b) Velocidad Simulación Hardware en Tiempo Real

Figura 3.13: Dinámica de Vuelo: Velocidad

correspondiente a la primera etapa del cohete. Simultáneamente, se inició el encendido del único propulsor de la segunda etapa, proporcionando una fuerza de empuje

constante durante aproximadamente **130** segundos con **250 kg** mas la masa del combustible de **2050 kg**, antes de su desacoplamiento de la carga útil. Finalmente, se llevó a cabo la maniobra destinada a conferir la velocidad necesaria a la carga útil de **150 kg**, permitiendo su inserción orbital a una altitud predeterminada, para lo cual se emplearon alrededor de **3250** segundos.

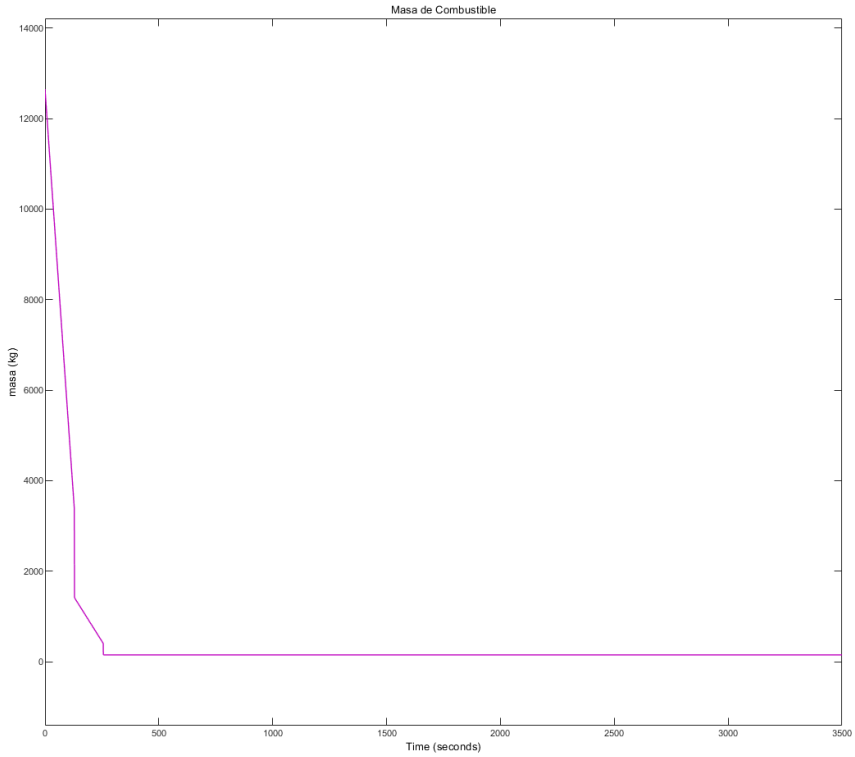
Tanto en la simulación convencional como en la simulación de Hardware In The Loop, se constató que el comportamiento de la masa del cohete, en función del tiempo y para cada una de las etapas, no se vio alterado al ser ejecutado en tiempo real (Fig. 3.14a, Fig. 3.14b).

En estas dos figuras, la pendiente de cada curva corresponde al flujo másico del motor del cohete. Con arreglo a los parámetros de diseño, se logró mantener constante el flujo másico al controlar la superficie quemada por el propelente a lo largo del tiempo, manteniendo así inalterada la fuerza de empuje del motor del cohete.

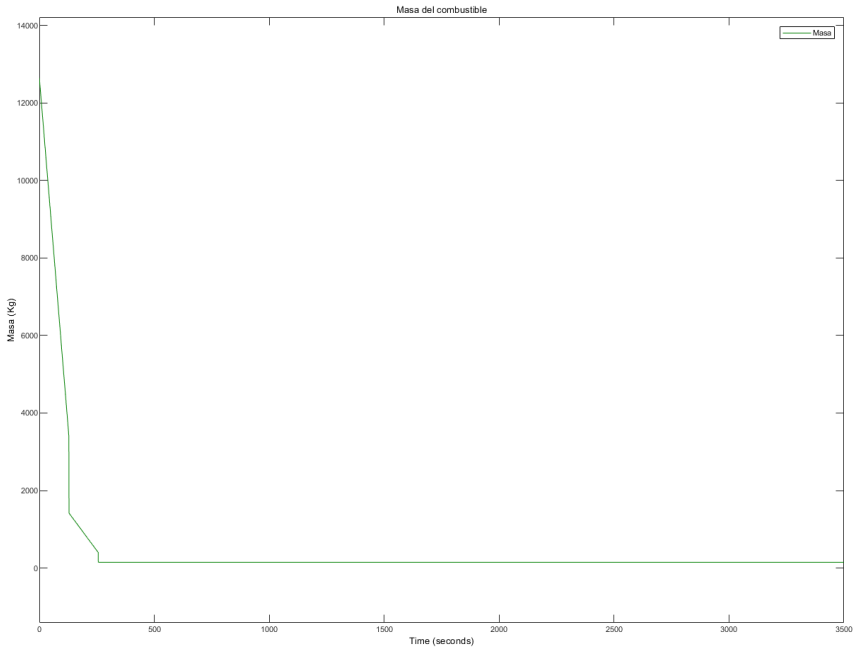
Las gráficas 3.15 presentan el comportamiento de los ángulos de Euler en las simulaciones convencional y de Hardware in the Loop (HIL). En la simulación convencional (Fig 3.13a), se logra estabilizar únicamente el ángulo que está siendo manipulado (Pitch) alrededor del cero, a diferencia de los otros ejes (Roll, Yaw). Por otro lado, en la figura correspondiente a los ángulos de Euler en la simulación de Hardware in the Loop (HIL) (Fig 3.13b), se aprecia que los dos ángulos previamente inestables se estabilizan alrededor del cero aproximadamente a los **2.000** segundos, momento en el cual la altitud alcanza su punto máximo y la velocidad disminuye notablemente. Esto se debe a que, al pertenecer a una simulación en tiempo real, el sistema utiliza cada una de sus componentes, a diferencia de la simulación convencional que solo se enfoca en un eje predeterminado.

No obstante, en la simulación convencional se evidencia el intento de estabilizar el ángulo Pitch alrededor de un grado. Aunque este valor es muy cercano a cero, dichos ajustes ocurren mientras el motor está en funcionamiento y pueden influir en la trayectoria del cohete. Durante este período se introducen fuerzas en los ejes del plano vertical, adicionales a las provocadas por la inclinación en el motor, que no se compensarán a lo largo del vuelo.

Por lo tanto, el comportamiento de los ángulos de Euler confirma la estabilidad del cohete y, al mismo tiempo, introduce pequeños valores de empuje adicionales en los ejes horizontales, lo que resulta en una contribución a un desplazamiento lateral.

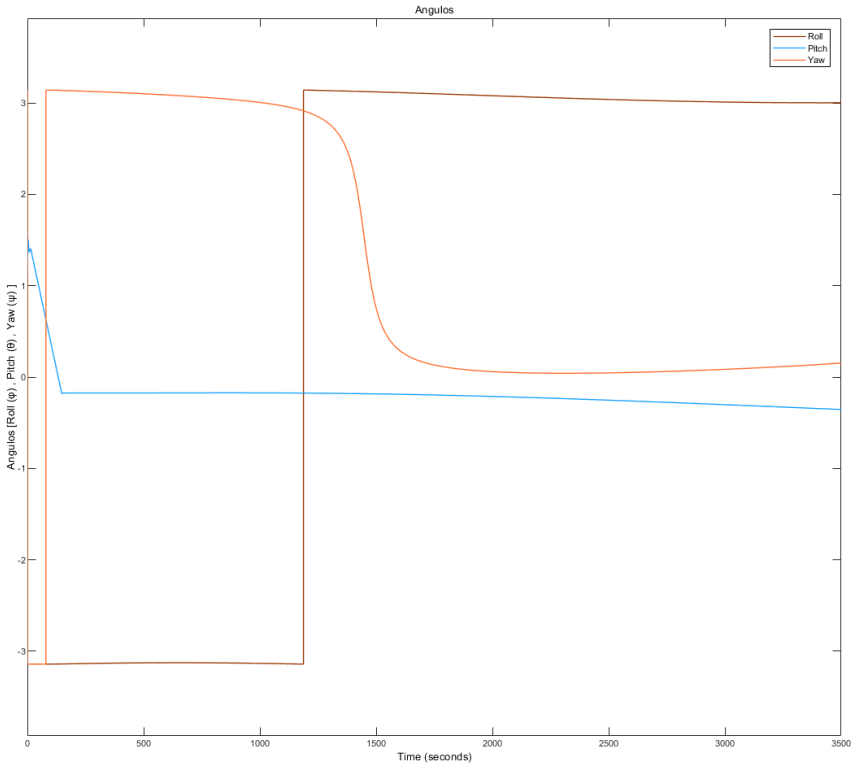


(a) Masa Carga Útil Simulación Convencional

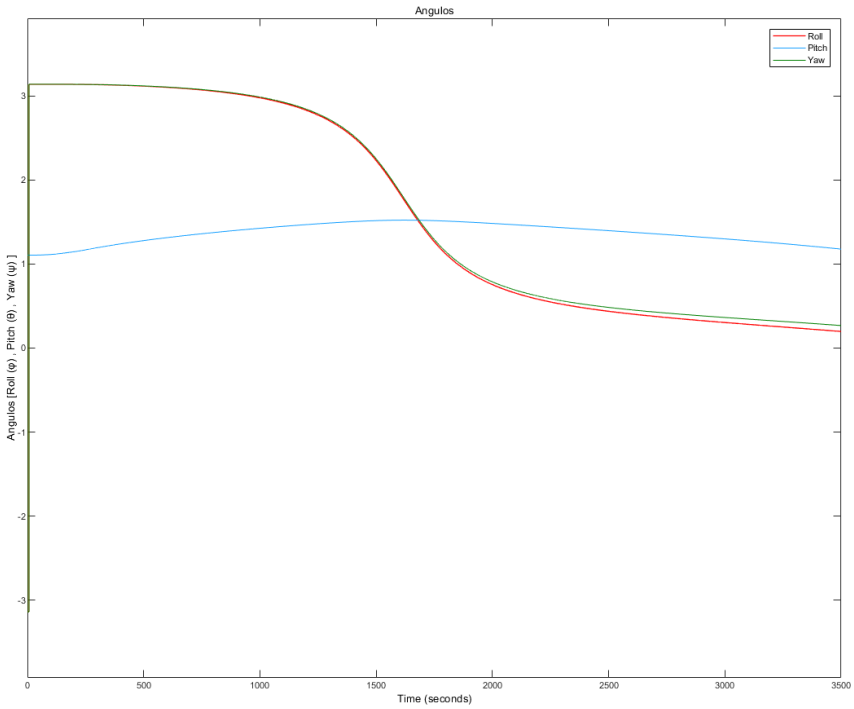


(b) Masa Carga Útil Simulación Hardware en Tiempo Real

Figura 3.14: Dinámica de Vuelo: Masa

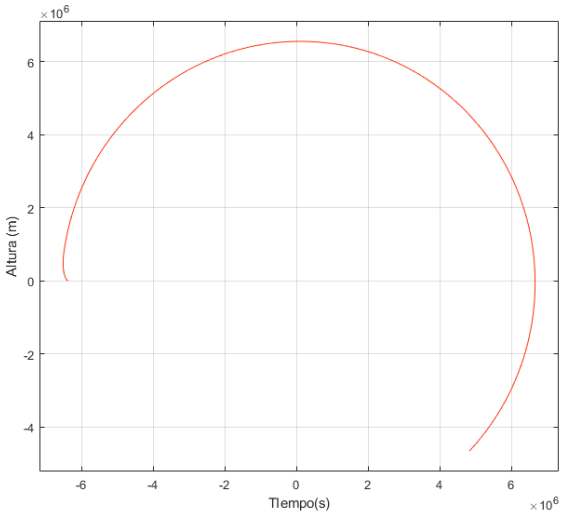


(a) Ángulos Simulación Convencional

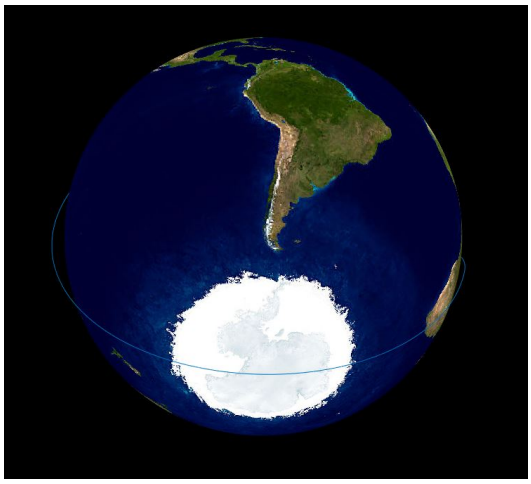


(b) Ángulos Simulación Hardware en Tiempo Real

Figura 3.15: Ángulos de Euler

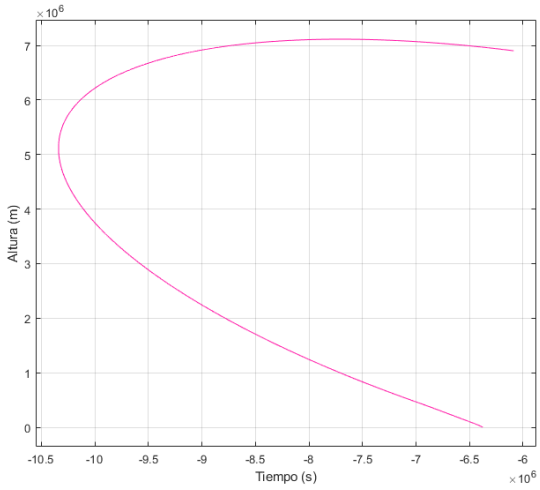


(a) Trayectoria Elipsoidal

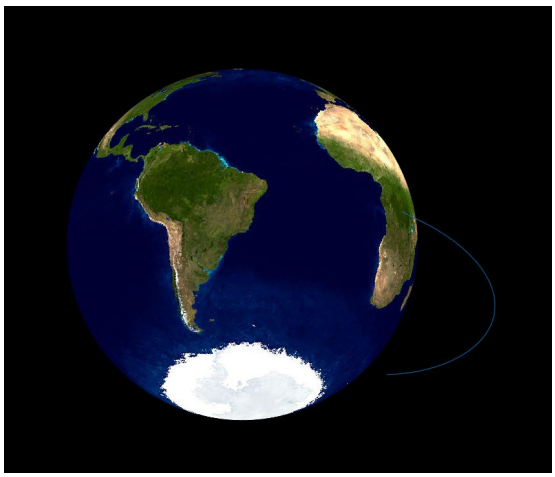


(b) Trayectoria Globo Terráqueo

Figura 3.16: Trayectoria Simulación Convencional



(a) Trayectoria Elipsoidal



(b) Trayectoria Globo Terráqueo

Figura 3.17: Trayectoria Simulación Hardware en Tiempo Real

La proyección orbital, derivada de la simulación tridimensional (Figuras 3.16b y 3.17b), sobre un plano bidimensional definido por latitud y longitud, resulta en las Figuras 3.16a y 3.17a. Estas representaciones gráficas proporcionan una descripción detallada de la trayectoria seguida por la carga útil en el sistema de coordenadas geocéntricas. Permiten determinar las áreas geográficas por las que transcurre el cuerpo en función del tiempo, lo cual es fundamental para la planificación estratégica de estaciones terrestres encargadas del control, monitoreo, transmisión y recepción de datos,

en concordancia con la naturaleza específica de la misión a realizar.

A lo largo del periodo de rotación alrededor del planeta, se observa una evolución de los elementos orbitales en función del tiempo, en contraposición a las simulaciones convencional y de Hardware in the Loop (HIL). Estos cambios indican las posibles perturbaciones debidas al achatamiento terrestre y a la resistencia aerodinámica de la atmósfera. Para este análisis, se ha considerado el intervalo de tiempo desde el momento en que el cohete se separa de la superficie terrestre hasta que alcanza su órbita designada, es decir, la duración total de la misión, que en el caso del HIL es de aproximadamente 20 minutos, mientras que en la simulación convencional es de 2 minutos. Durante este periodo, se observa claramente cómo la simulación HIL entra en una zona suborbital debido a su mayor altitud de apogeo en comparación con la simulación convencional, que logra estabilizarse en la órbita prevista.

3.7. Ejecutar Sistema de Dirección con Motores

Conforme se avanza en el proceso de investigación, cobra importancia la consideración de su implementación o continuación. Es por ello que, con el propósito de tener un primer acercamiento a la realidad y profundizar en el entendimiento del comportamiento, se ha determinado emplear un controlador difuso simple para gestionar los servo motores. Estos simularán la respuesta de las aletas ante las lecturas proporcionadas por el sensor MPU6050, el cual suministrará datos de ángulos en sus respectivos ejes X,Y,Z (Roll, Pitch, Yaw).

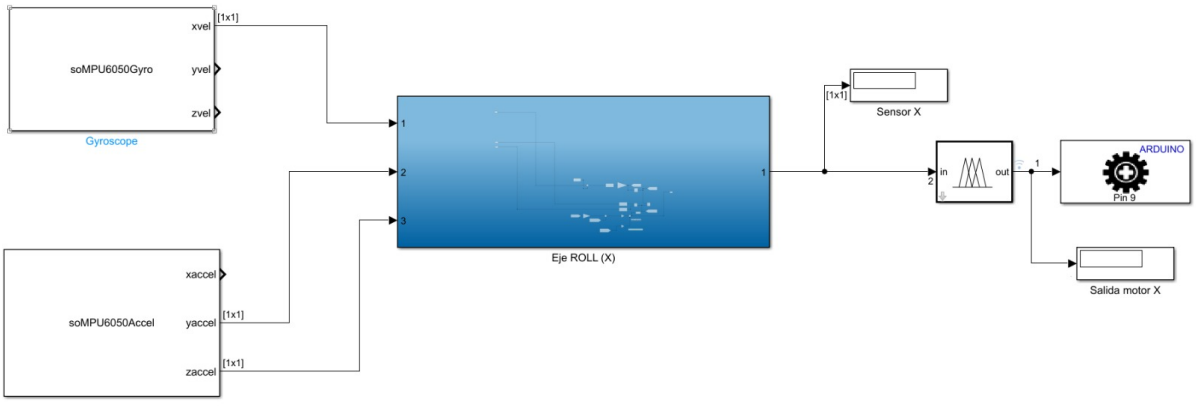


Figura 3.18: Esquemático Dirección de Motores

Tal como se ha mencionado con antelación, Simulink presenta una serie de bloques que facilitan la interacción con el usuario en diseños como este. A continuación, en la tabla 3.8 se detallan los bloques que permiten llevar a cabo la implementación del esquemático representado en la figura 3.18.

Dentro del amplio conjunto de herramientas de MATLAB, se encuentra el conjunto de herramientas de Fuzzy Logic Designer, el cual resulta de gran utilidad en el desarrollo de lógica difusa.

La base de conocimiento del Controlador de Logica Difusa (FCL) se compone de dos elementos fundamentales: una base de datos y una base de reglas. La base de datos proporciona las definiciones necesarias para la formulación de reglas de control y la manipulación de datos difusos. El número máximo de reglas está determinado por el producto de las particiones de todas las variables lingüísticas que ingresan al FCL. Para hablar de una base de reglas, es esencial elegir cuáles variables se considerarán como entradas y cuáles como salidas en el FCL.

En esta implementación, las variables lingüísticas de entrada para el Controlador de Lógica Difusa (FCL) son las siguientes:

Entrada	**Salida**
Ángulos MPU6050	Posición Servo
INPUT_X	OUTPUT_X

Tabla 3.7: Variables Lingüísticas

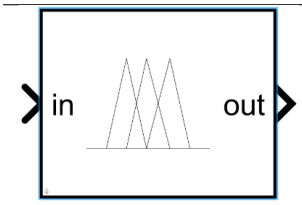
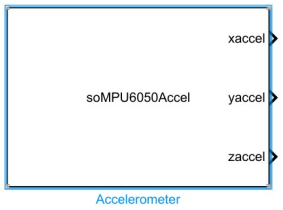
Bloques	Características
	El bloque Fuzzy Logic Controller implementa un sistema de inferencia difusa (FIS) en Simulink.
	Acelerómetro MPU6050. Se emplea esta clase para obtener la lectura del estado de la aceleración
	Giroscopio MPU6050. Modo DLP = 0,1,2,3,4,5,6 correspondiente a un filtro de paso bajo con ancho de banda 256, 188, 98, 42, 20, 10, 5 Hz con retardo 0.98, 1.9, 2.8, 4.8, 8.3, 13.4, 18.6 ms.
	Ajusta la posición del eje del servomotor a un ángulo determinado según el valor de entrada enviado al bloque en el pin de hardware de Arduino.

Tabla 3.8: Bloques de Simulink

Donde los universos de discurso de estas variables, son:

Entrada	Salida
[0 200]	[0 180]

Tabla 3.9: Universos de Discurso

Entendido esto, se configuró en Fuzzy Logic Designer la cantidad de entradas y salidas que se emplean para este controlador, tal como se ilustra en la figura 3.19.

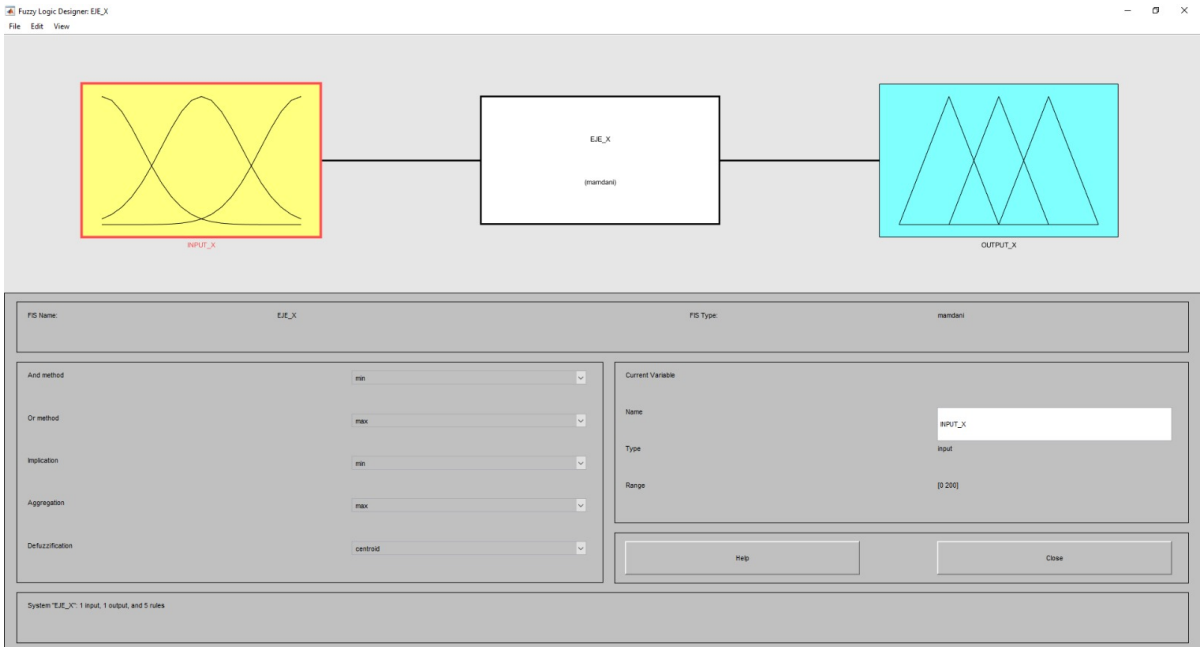


Figura 3.19: Fuzzy Logic Controller

Se emplearon un total de cinco valores lingüísticos para cada una de estas variables.

Angulo	Entrada	Posición	Salida
-90	INP_FULL_IZQ_X	0	OUT_FULL_IZQ_X
-45	INP_IZQ_X	45	OUT_IZQ_X
0	INP_CENTER_X	90	OUT_CENTER_X
45	INP_DER_X	135	OUT_DER_X
90	INP_FULL_DER_X	180	OUT_FULL_DER_X

Tabla 3.10: Valores Lingüísticos

Una vez que se determinaron los valores lingüísticos, se establecieron las funciones de membresía tanto para las entradas (Fig 3.20) como para las salidas (Fig 3.21).

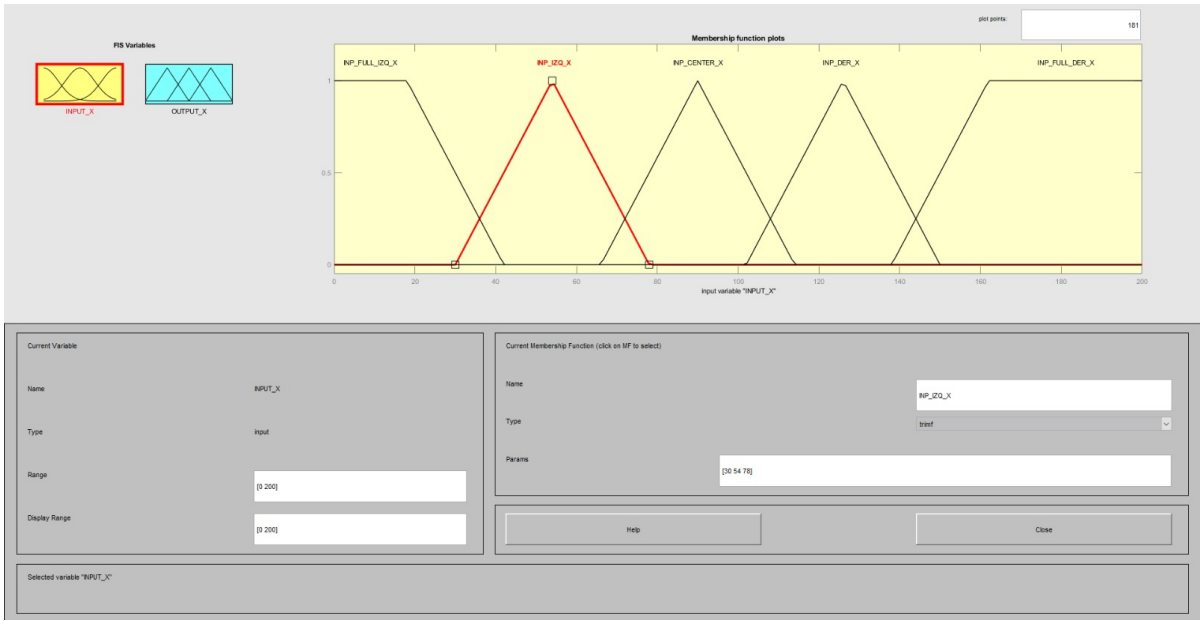


Figura 3.20: Variables de Entrada

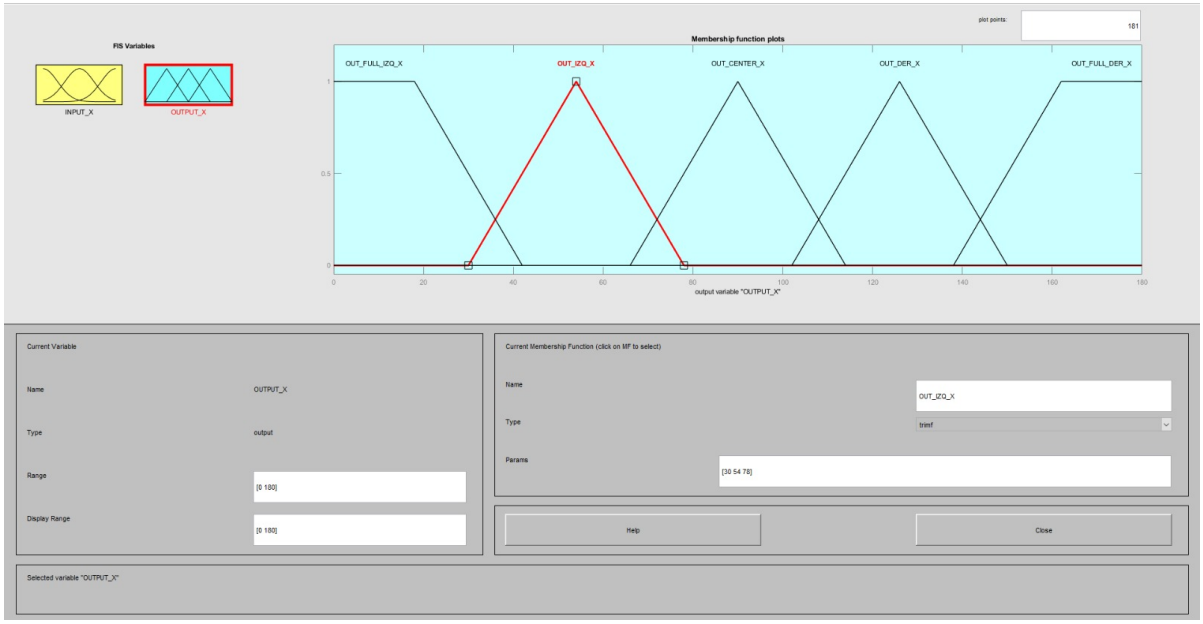


Figura 3.21: Variables de Salida

Es importante destacar que, en un inicio, se desarrolló el controlador únicamente para el eje Roll (X). Sin embargo, el procedimiento es análogo para los distintos ejes.

Para llevar a cabo el proceso con respecto al eje X, fue necesario colocar el sensor en la posición adecuada, utilizando como guía la figura 2.12.

Después de establecer las funciones de membresía, se procedió con la formulación de las reglas difusas. En total, se generaron 5 reglas, las cuales están detalladas en la tabla 3.11.

Reglas	**-90**	**-45**	**0**	**45**	**90**
-90	0	45	90	135	180
-45	45	0	45	90	135
0	90	45	0	45	90
45	135	90	45	0	45
90	180	135	90	45	0

Tabla 3.11: Reglas Difusas

Una vez que se implementaron estas reglas en Fuzzy Logic Designer, podíamos condicionar las entradas para obtener la salida correspondiente (Fig 3.22).

Además, el diseño de este controlador fue el resultado de un largo proceso experimental que incluyó pruebas de funcionamiento, adaptaciones y ajustes de variables lingüísticas, así como la definición de rangos de universos de discurso, entre otros aspectos.

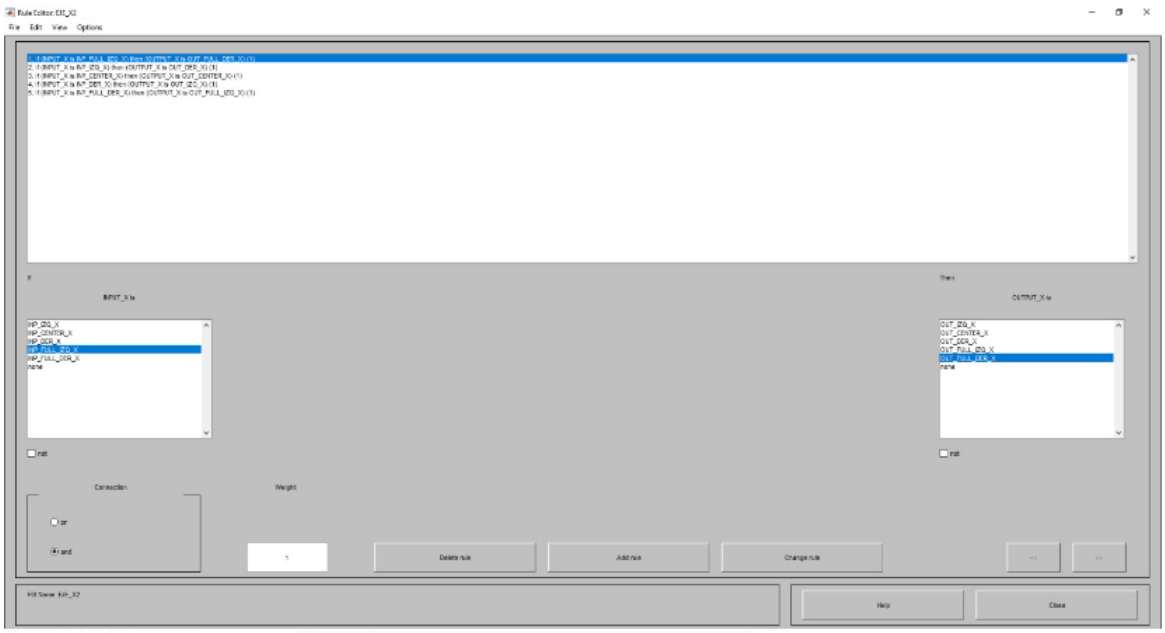


Figura 3.22: Reglas Lingüísticas

Por lo tanto, la entrada puede ser ajustada de manera flexible para observar la salida defuzzificada en diferentes estados (Fig 3.23).

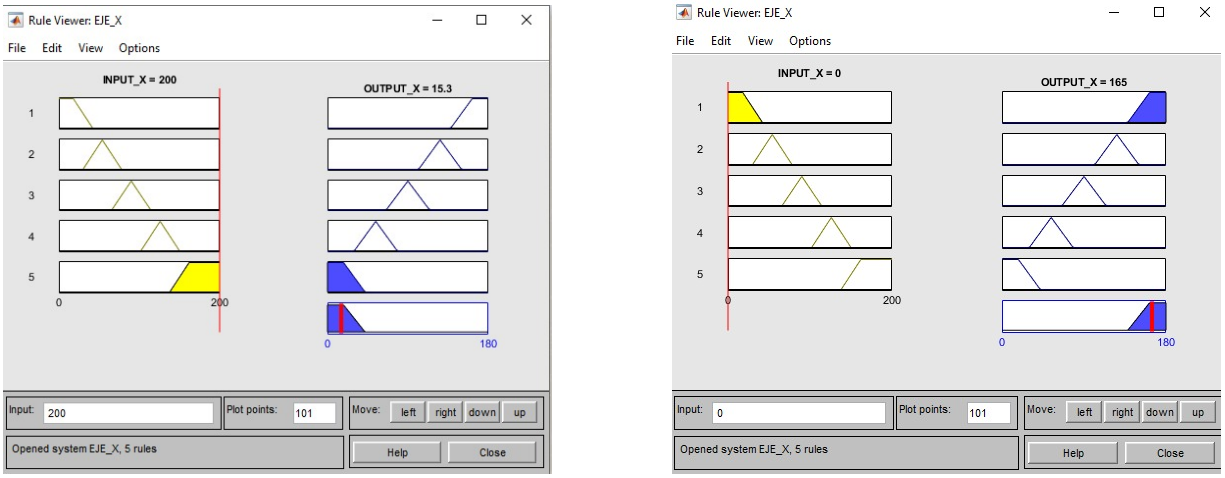


Figura 3.23: Reglas y Defuzzificación

Es relevante señalar que había un desfase de 15.3 grados. Por esta razón, en la implementación se decidió incluir un filtro complementario, lo que permitió minimizar considerablemente dicho desfase.

Capítulo 4

Conclusiones y Trabajos Futuros

En este estudio, se ha llevado a cabo un análisis detallado de la trayectoria de vuelo del cohete **Electron** de la empresa **RocketLab** el cual consta de dos etapas. Este análisis se basó en ecuaciones diferenciales que describen el movimiento del **Electron** en un sistema de referencia no inercial, incorporando las características de diseño del vehículo y parámetros ambientales como la atmósfera del planeta. Estos elementos se integraron en un modelo implementado en Simulink/MATLAB R2023a, lo que permitió simular la trayectoria de vuelo y determinar los rangos de misión aplicables en la práctica. Aunque el trabajo no ha sido validado experimentalmente, se espera que en futuras aplicaciones y estudios, los datos puedan ser corroborados experimentalmente, contribuyendo así al avance en el ámbito espacial del país.

La combinación de simulaciones convencionales con pruebas de hardware en tiempo real (HILS) se ha considerado como un enfoque válido y efectivo para evaluar el rendimiento del cohete en condiciones simuladas y reales, demostrando su aplicabilidad en el ámbito aeroespacial. Esta metodología se erige como una herramienta indispensable en el proceso de diseño y desarrollo, permitiendo una comprensión detallada y óptima del rendimiento en diversas fases de vuelo.

El análisis tuvo en cuenta las ecuaciones que gobiernan el movimiento del cohete, como se detalla en el **capítulo 2**. Estas ecuaciones, incorporadas en el modelo, posibilitaron la predicción del comportamiento de la trayectoria de vuelo del **Electron** y el desempeño de cada una de sus etapas.

Las simulaciones realizadas revelan que el **Electron** tiene la capacidad de transportar una carga de **950 kg** en su primera etapa y **250 kg** en su segunda etapa. Sin embargo, al estabilizarse en órbita, su carga útil se reduce a **150 kg**, excluyendo la masa del combustible. Se concluye ademas, que la simulación **Hardware In The Loop Simulation** logra un vuelo suborbital, donde el motor se enciende aproximadamente veinte minutos mientras alcanza su altura máxima para llegar a la orbita deseada. Por el contrario, en un vuelo orbital que corresponde a la simulación convencional, el cohete acelera durante aproximadamente dos minutos y alcanza una velocidad su-

ficientemente alta como para mantener su órbita. Sin embargo, es importante tener en consideración que la duración de estos minutos de simulación depende del rendimiento del equipo de procesamiento que se utilice, para esta investigación se utilizó un portátil con las siguientes especificaciones: MSI sword 15, core i7 11 generación, tarjeta gráfica gtx 3050 ti 6g, 16 g ram, 1 tera + 512 de disco duro.

El rendimiento aerodinámico establece las cargas que la estructura deberá soportar debido a las fuerzas aerodinámicas experimentadas durante la fase de ascenso atmosférico. Estas fuerzas se ven influenciadas por la presión dinámica, que cambia en relación a la densidad en función de la altitud, así como por la velocidad impartida al vehículo por los motores del **Electron**.

Basándonos en la experiencia adquirida a través del control difuso en la operación real, se concluye que el control difuso puede ser una forma práctica y efectiva de aumentar el nivel de control coordinativo en procesos aeroespaciales multivariables. Se recomienda implementar el HILS con el mismo modelado y un controlador difuso para realizar comparaciones con el PID. Además, en futuras implementaciones se sugiere utilizar una plataforma embebida de mayor potencia para mantener una alta velocidad de transmisión sin pérdida de información y así, evitar la saturación de la memoria del microcontrolador.

Capítulo 5

Anexos

5.0.1. Software y Hardware Utilizado

- Matlab
- Simulink
- Arduino

5.0.2. Metodología y Cronograma de Actividades

Actividad	Ago 2022	Sep 2022	Oct 2022	Nov 2022	Dic 2022	Ene 2023	Feb 2023	Mar 2023	May 2023	Jun 2023	Jul 2023
1											
2											
3											
4											
5											
6											
7											
8											
9											

Actividades:

1. Recolectar información para identificar un modelo acorde.
2. Identificar el sistema de control especifico que se adapten a las necesidades.
3. Se afianzarán conocimientos en el campo de la mecánica (dinámica y estática) de vuelo de cohetes a escala.

4. Implementar estrategias e incorporar el modelo matemático con el control correspondiente.
5. Establecer las características del microcontrolador y definir las interfaces necesarias.
6. Asociar los parámetros acordes con la interfaz y el microcontrolador
7. Diseño e implementación del sistema Hardware In The Loop Simulation.
8. Basándose en el modelo matemático y comportamiento físico de un cohete a escala, se determinará la mecánica de vuelo dentro del simulador.
9. Con ayuda del Hardware In The Loop Simulation se validará un sistema de control especifico que tiene como finalidad visualizar el comportamiento esperado del cohete a escala.

5.0.3. Presupuesto Investigación

Item	Rubro	Estudiantes	Universidad	Total
1	Personal	$2'160.000	$2'300.000	$4'460.000
2	Equipos	$1'200.000	$1'500.000	$2'700.000
3	Software	$0.00	$16'000.000	$16'000.000
4	Total	$3'360.000	$19'800.000	$23'160.000

Nota: Las estudiantes asumieron los costos de lo que conllevaba diseñar e implementar el trabajo de investigación.

5.0.4. Códigos Programa MATLAB

Código controlador PID:

```
1 *
2  * Academic License - for use in teaching, academic research,
       and meeting
3  * course requirements at degree granting institutions only.
       Not for
4  * government, commercial, or other organizational use.
5  *
6  * File: Controlador.c
7  *
8  * Code generated for Simulink model 'Controlador'.
9  *
```

```
10  * Model version                  : 1.11
11  * Simulink Coder version         : 9.9 (R2023a) 19-Nov-2022
12  * C/C++ source code generated on : Sun Sep 24 19:40:22 2023
13  *
14  * Target selection: ert.tlc
15  * Embedded hardware selection: Atmel->AVR
16  * Code generation objective: Execution efficiency
17  * Validation result: Not run
18  */
19
20 #include "Controlador.h"
21 #include "rtwtypes.h"
22 #include "Controlador_private.h"
23 #include <math.h>
24
25 /* Continuous states */
26 X_Controlador_T Controlador_X;
27
28 /* Block signals and states (default storage) */
29 DW_Controlador_T Controlador_DW;
30
31 /* Real-time model */
32 static RT_MODEL_Controlador_T Controlador_M_;
33 RT_MODEL_Controlador_T *const Controlador_M = &Controlador_M_
       ;
34
35 /*
36  * This function updates continuous states using the ODE3
        fixed-step
37  * solver algorithm
38  */
39 static void rt_ertODEUpdateContinuousStates(RTWSolverInfo *si
       )
40 {
41   /* Solver Matrices */
42   static const real_T rt_ODE3_A[3] = {
43     1.0/2.0, 3.0/4.0, 1.0
44   };
45
46   static const real_T rt_ODE3_B[3][3] = {
47     { 1.0/2.0, 0.0, 0.0 },
48   { 0.0, 3.0/4.0, 0.0 },
49
50     { 2.0/9.0, 1.0/3.0, 4.0/9.0 }
```

```
51  };
52
53  time_T t = rtsiGetT(si);
54  time_T tnew = rtsiGetSolverStopTime(si);
55  time_T h = rtsiGetStepSize(si);
56  real_T *x = rtsiGetContStates(si);
57  ODE3_IntgData *id = (ODE3_IntgData *)rtsiGetSolverData(si);
58  real_T *y = id->y;
59  real_T *f0 = id->f[0];
60  real_T *f1 = id->f[1];
61  real_T *f2 = id->f[2];
62  real_T hB[3];
63  int_T i;
64  int_T nXc = 2;
65  rtsiSetSimTimeStep(si,MINOR_TIME_STEP);
66  /* Save the state values at time t in y, we'll use x as
       ynew. */
67  (void) memcpy(y, x,
68                (uint_T)nXc*sizeof(real_T));
69
70  /* Assumes that rtsiSetT and ModelOutputs are up-to-date */
71  /* f0 = f(t,y) */
72  rtsiSetdX(si, f0);
73  Controlador_derivatives();
74
75  /* f(:,2) = feval(odefile, t + hA(1), y + f*hB(:,1), args
       (:)(*)); */
76  hB[0] = h * rt_ODE3_B[0][0];
77  for (i = 0; i < nXc; i++) {
78    x[i] = y[i] + (f0[i]*hB[0]);
79  }
80
81  rtsiSetT(si, t + h*rt_ODE3_A[0]);
82  rtsiSetdX(si, f1);
83  Controlador_step();
84  Controlador_derivatives();
85
86  /* f(:,3) = feval(odefile, t + hA(2), y + f*hB(:,2), args
       (:)(*)); */
87  for (i = 0; i <= 1; i++) {
88    hB[i] = h * rt_ODE3_B[1][i];
89  }
90
91  for (i = 0; i < nXc; i++) {
```

```
92      x[i] = y[i] + (f0[i]*hB[0] + f1[i]*hB[1]);
93    }
94
95    rtsiSetT(si, t + h*rt_ODE3_A[1]);
96    rtsiSetdX(si, f2);
97    Controlador_step();
98    Controlador_derivatives();
99
100   /* tnew = t + hA(3);
101      ynew = y + f*hB(:,3); */
102   for (i = 0; i <= 2; i++) {
103     hB[i] = h * rt_ODE3_B[2][i];
104   }
105
106
107   for (i = 0; i < nXc; i++) {
108     x[i] = y[i] + (f0[i]*hB[0] + f1[i]*hB[1] + f2[i]*hB[2]);
109   }
110
111   rtsiSetT(si, tnew);
112   rtsiSetSimTimeStep(si,MAJOR_TIME_STEP);
113 }
114
115 real_T rt_roundd(real_T u)
116 {
117   real_T y;
118   if (fabs(u) < 4.503599627370496E+15) {
119     if (u >= 0.5) {
120       y = floor(u + 0.5);
121     } else if (u > -0.5) {
122       y = 0.0;
123     } else {
124       y = ceil(u - 0.5);
125     }
126   } else {
127     y = u;
128   }
129
130   return y;
131 }
132
133 /* Model step function */
134 void Controlador_step(void)
135 {
```

```
136   real_T dataIn[3];
137   real_T rtb_Estrategiadecontrol;
138   int16_T i;
139   char_T labelTerminated[13];
140   uint8_T b_status;
141   static const char_T tmp[13] = "GPS_latitude";
142   if (rtmIsMajorTimeStep(Controlador_M)) {
143     /* set solver stop time */
144     rtsiSetSolverStopTime(&Controlador_M->solverInfo,
145                           ((Controlador_M->Timing.clockTick0
                                +1)*
146       Controlador_M->Timing.stepSize0));
147   }                                     /* end MajorTimeStep
       */
148   /* Update absolute time of base rate at minor time step */
149   if (rtmIsMinorTimeStep(Controlador_M)) {
150     Controlador_M->Timing.t[0] = rtsiGetT(&Controlador_M->
        solverInfo);
151   }
152
153   if (rtmIsMajorTimeStep(Controlador_M)) {
154     /* MATLABSystem: '<Root>/Serial Receive' */
155     if (Controlador_DW.obj_m.Protocol != 0.0) {
156       Controlador_DW.obj_m.Protocol = 0.0;
157     }
158
159     if (Controlador_DW.obj_m.QueueSizeFactor != 3.0) {
160       Controlador_DW.obj_m.QueueSizeFactor = 3.0;
161     }
162
163     MW_Serial_read(1, Controlador_DW.obj_m.DataSizeInBytes,
164                    &rtb_Estrategiadecontrol, &b_status);
165
166     /* Gain: '<S3>/Gain' incorporates:
167      *  MATLABSystem: '<Root>/Serial Receive'
168      */
169     Controlador_DW.Gain = 57.295779513082323 *
        rtb_Estrategiadecontrol;
170   }
171
172   /* Step: '<S4>/Step' */
173   if (Controlador_M->Timing.t[0] < 15.0) {
174     rtb_Estrategiadecontrol = 0.0;
175   } else {
```

```
176     rtb_Estrategiadecontrol = 0.0085;
177   }
178
179   /* Sum: '<S1>/Sum1' incorporates:
180    *  Clock: '<S4>/Clock'
181    *  Constant: '<S1>/Angulo de picada'
182    *  Constant: '<S4>/Constant'
183    *  Product: '<S1>/Product'
184    *  Product: '<S4>/Product'
185    *  Step: '<S4>/Step'
186    *  Sum: '<S4>/Sum'
187    */
188   rtb_Estrategiadecontrol = 80.0 - (Controlador_M->Timing.t
         [0] - 15.0) *
189     rtb_Estrategiadecontrol * 80.0;
190
191   /* Switch: '<S1>/Switch' */
192   if (rtb_Estrategiadecontrol < 10.0) {
193     rtb_Estrategiadecontrol = 10.0;
194   }
195    /* End of Switch: '<S1>/Switch' */
196
197   /* Gain: '<S32>/Derivative Gain' incorporates:
198    *  Gain: '<S43>/Proportional Gain'
199    *  Sum: '<S2>/Sum3'
200    *  Switch: '<S1>/Switch1'
201    */
202   rtb_Estrategiadecontrol = (Controlador_DW.Gain -
         rtb_Estrategiadecontrol) *
203     0.01;
204
205   /* Gain: '<S41>/Filter Coefficient' incorporates:
206    *  Gain: '<S32>/Derivative Gain'
207    *  Integrator: '<S33>/Filter'
208    *  Sum: '<S33>/SumD'
209    */
210   Controlador_DW.FilterCoefficient = (rtb_Estrategiadecontrol
         -
211     Controlador_X.Filter_CSTATE) * 100.0;
212
213   /* Switch: '<S1>/Switch1' incorporates:
214    *  Constant: '<S1>/Constante para saer el tiempo de
         desactivacion del sistema. no se curviara '
215    *  Integrator: '<S1>/Integrator'
```

```
216     *  Sum: '<S1>/Sum'
217     */
218   if (250.0 - Controlador_X.Integrator_CSTATE > 0.0) {
219     /* MATLABSystem: '<Root>/Serial Transmit' incorporates:
220      *  Constant: '<S1>/Cte nula'
221      *  Sum: '<S47>/Sum'
222      */
223     dataIn[0] = 0.0;
224     dataIn[1] = rtb_Estrategiadecontrol + Controlador_DW.
          FilterCoefficient;
225     dataIn[2] = 0.0;
226   } else {
227     /* MATLABSystem: '<Root>/Serial Transmit' incorporates:
228      *  Constant: '<S1>/Vector nulo. '
229      */
230     dataIn[0] = 0.0;
231     dataIn[1] = 0.0;
232     dataIn[2] = 0.0;
233   }
234
235   /* MATLABSystem: '<Root>/Serial Transmit' */
236   if (Controlador_DW.obj.Protocol != 0.0) {
237     Controlador_DW.obj.Protocol = 0.0;
238   }
239
240   for (i = 0; i < 13; i++) {
241     labelTerminated[i] = tmp[i];
242   }
243
244   MW_Serial_write(Controlador_DW.obj.port, &dataIn[0], 3.0,
245                   Controlador_DW.obj.dataSizeInBytes,
246                   Controlador_DW.obj.sendModeEnum,
                          Controlador_DW.obj.dataType,
247                   Controlador_DW.obj.sendFormatEnum, 5.0, &
                          labelTerminated[0]);
248   if (rtmIsMajorTimeStep(Controlador_M)) {
249     rt_ertODEUpdateContinuousStates(&Controlador_M->
          solverInfo);
250
251     /* Update absolute time for base rate */
252     /* The "clockTick0" counts the number of times the code
          of this task has
253      * been executed. The absolute time is the multiplication
            of "clockTick0"
```

```
254       * and "Timing.stepSize0". Size of "clockTick0" ensures
              timer will not
255       * overflow during the application lifespan selected.
256       */
257      ++Controlador_M->Timing.clockTick0;
258      Controlador_M->Timing.t[0] = rtsiGetSolverStopTime
259        (&Controlador_M->solverInfo);
260
261      {
262      /* Update absolute timer for sample time: [0.1s, 0.0s] */
263        /* The "clockTick1" counts the number of times the code
              of this task has
264         * been executed. The resolution of this integer timer
              is 0.1, which is the step size
265         * of the task. Size of "clockTick1" ensures timer will
               not overflow during the
266         * application lifespan selected.
267         */
268        Controlador_M->Timing.clockTick1++;
269      }
270    }                                   /* end MajorTimeStep
         */
271  }
272
273  /* Derivatives for root system: '<Root>' */
274  void Controlador_derivatives(void)
275  {
276    XDot_Controlador_T *_rtXdot;
277    _rtXdot = ((XDot_Controlador_T *) Controlador_M->derivs);
278
279    /* Derivatives for Integrator: '<S33>/Filter' */
280    _rtXdot->Filter_CSTATE = Controlador_DW.FilterCoefficient;
281
282    /* Derivatives for Integrator: '<S1>/Integrator'
         incorporates:
283     *  Constant: '<S1>/Constant'
284     */
285    _rtXdot->Integrator_CSTATE = 1.0;
286  }
287
288  /* Model initialize function */
289  void Controlador_initialize(void)
290  {
291    /* Registration code */
```

```
292 {
293   /* Setup solver object */
294   rtsiSetSimTimeStepPtr(&Controlador_M->solverInfo,
295                         &Controlador_M->Timing.simTimeStep)
                              ;
296   rtsiSetTPtr(&Controlador_M->solverInfo, &rtmGetTPtr(
        Controlador_M));
297   rtsiSetStepSizePtr(&Controlador_M->solverInfo,
298                      &Controlador_M->Timing.stepSize0);
299   rtsiSetdXPtr(&Controlador_M->solverInfo, &Controlador_M->
        derivs);
300   rtsiSetContStatesPtr(&Controlador_M->solverInfo, (real_T
        **)
301                        &Controlador_M->contStates);
302   rtsiSetNumContStatesPtr(&Controlador_M->solverInfo,
303     &Controlador_M->Sizes.numContStates);
304   rtsiSetNumPeriodicContStatesPtr(&Controlador_M->
        solverInfo,
305     &Controlador_M->Sizes.numPeriodicContStates);
306   rtsiSetPeriodicContStateIndicesPtr(&Controlador_M->
        solverInfo,
307     &Controlador_M->periodicContStateIndices);
308   rtsiSetPeriodicContStateRangesPtr(&Controlador_M->
        solverInfo,
309     &Controlador_M->periodicContStateRanges);
310   rtsiSetErrorStatusPtr(&Controlador_M->solverInfo, (&
        rtmGetErrorStatus
311     (Controlador_M)));
312   rtsiSetRTModelPtr(&Controlador_M->solverInfo,
        Controlador_M);
313 }
314
315 rtsiSetSimTimeStep(&Controlador_M->solverInfo,
      MAJOR_TIME_STEP);
316 Controlador_M->intgData.y = Controlador_M->odeY;
317 Controlador_M->intgData.f[0] = Controlador_M->odeF[0];
318 Controlador_M->intgData.f[1] = Controlador_M->odeF[1];
319 Controlador_M->intgData.f[2] = Controlador_M->odeF[2];
320 Controlador_M->contStates = ((X_Controlador_T *) &
      Controlador_X);
321 rtsiSetSolverData(&Controlador_M->solverInfo, (void *)&
      Controlador_M->intgData);
322 rtsiSetIsMinorTimeStepWithModeChange(&Controlador_M->
      solverInfo, false);
```

```
323   rtsiSetSolverName(&Controlador_M->solverInfo,"ode3");
324   rtmSetTPtr(Controlador_M, &Controlador_M->Timing.tArray[0])
        ;
325   Controlador_M->Timing.stepSize0 = 0.1;
326
327   {
328     real_T tmp;
329     uint8_T tmp_0;
330
331     /* InitializeConditions for Integrator: '<S33>/Filter' */
332     Controlador_X.Filter_CSTATE = 0.0;
333
334     /* InitializeConditions for Integrator: '<S1>/Integrator'
          */
335     Controlador_X.Integrator_CSTATE = 0.0;
336
337     /* Start for MATLABSystem: '<Root>/Serial Receive' */
338     Controlador_DW.obj_m.matlabCodegenIsDeleted = false;
339     Controlador_DW.obj_m.Protocol = 0.0;
340     Controlador_DW.obj_m.QueueSizeFactor = 3.0;
341     Controlador_DW.obj_m.isInitialized = 1L;
342     Controlador_DW.obj_m.DataTypeWidth = 4U;
343     Controlador_DW.obj_m.DataSizeInBytes = Controlador_DW.
        obj_m.DataTypeWidth;
344     MW_SCI_Open(1);
345     Controlador_DW.obj_m.isSetupComplete = true;
346
347     /* Start for MATLABSystem: '<Root>/Serial Transmit' */
348     Controlador_DW.obj.matlabCodegenIsDeleted = false;
349     Controlador_DW.obj.Protocol = 0.0;
350     Controlador_DW.obj.isInitialized = 1L;
351     Controlador_DW.obj.port = 1.0;
352     Controlador_DW.obj.dataSizeInBytes = 8.0;
353     Controlador_DW.obj.dataType = 6.0;
354     Controlador_DW.obj.sendModeEnum = 0.0;
355     Controlador_DW.obj.sendFormatEnum = 0.0;
356     tmp = rt_roundd(Controlador_DW.obj.port);
357     if (tmp < 256.0) {
358       if (tmp >= 0.0) {
359         tmp_0 = (uint8_T)tmp;
360       } else {
361         tmp_0 = 0U;
362       }
363     } else {
```

```
364         tmp_0 = MAX_uint8_T;
365       }
366
367       MW_SCI_Open(tmp_0);
368       Controlador_DW.obj.isSetupComplete = true;
369
370       /* End of Start for MATLABSystem: '<Root>/Serial Transmit
             ' */
371     }
372   }
373
374   /* Model terminate function */
375   void Controlador_terminate(void)
376   {
377     /* Terminate for MATLABSystem: '<Root>/Serial Receive' */
378     if (!Controlador_DW.obj_m.matlabCodegenIsDeleted) {
379       Controlador_DW.obj_m.matlabCodegenIsDeleted = true;
380     }
381
382     /* End of Terminate for MATLABSystem: '<Root>/Serial
           Receive' */
383
384     /* Terminate for MATLABSystem: '<Root>/Serial Transmit' */
385     if (!Controlador_DW.obj.matlabCodegenIsDeleted) {
386       Controlador_DW.obj.matlabCodegenIsDeleted = true;
387     }
388
389     /* End of Terminate for MATLABSystem: '<Root>/Serial
           Transmit' */
390   }
391
392   /*
393    * File trailer for generated code.
394    *
395    * [EOF]
396    */
```

Código en C++ Programa Earth 3D:

```
1  %% Textured 3D Earth example
2  %
3  % Ryan Gray
4  % 8 Sep 2004
```

```
5  % Revised 9 March 2006, 31 Jan 2006, 16 Oct 2013
6
7  %% Options
8
9  space_color = 'k';
10 npanels = 180;   % Number of globe panels around the equator
       deg/panel = 360/npanels
11 alpha   = 1; % globe transparency level, 1 = opaque, through
       0 = invisible
12 %GMST0 = []; % Don't set up rotatable globe (ECEF)
13 GMST0 = 5.89496121282306; % Set up a rotatable globe at J2000
       .0
14
15
16 % Earth texture image
17 % Anything imread() will handle, but needs to be a 2:1
       unprojected globe
18 % image.
19
20 image_file = 'http://upload.wikimedia.org/wikipedia/commons/
       thumb/c/cd/Land_ocean_ice_2048.jpg/1024px-
       Land_ocean_ice_2048.jpg';
21
22 % Mean spherical earth
23
24 erad    = 6371008.7714; % equatorial radius (meters)
25 prad    = 6371008.7714; % polar radius (meters)
26 erot    = 7.2921158553e-5; % earth rotation rate (radians/sec
       )
27
28 %% Create figure
29
30 figure('Color', space_color);
31
32 hold on;
33
34 % Turn off the normal axes
35
36 set(gca, 'NextPlot','add', 'Visible','off');
37
38 axis equal;
39 axis auto;
40
41 % Set initial view
```

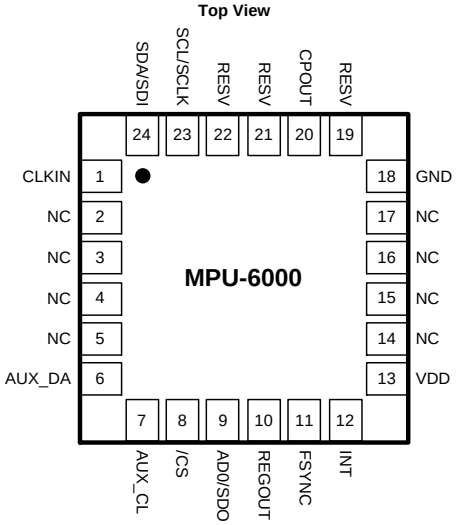
```
42
43 view(0,30);
44
45 axis vis3d;
46
47 %% Create wireframe globe
48
49 % Create a 3D meshgrid of the sphere points using the
       ellipsoid function
50
51 [x, y, z] = ellipsoid(0, 0, 0, erad, erad, prad, npanels);
52
53 globe = surf(x, y, -z, 'FaceColor', 'none', 'EdgeColor',
       0.5*[1 1 1]);
54
55 if ~isempty(GMST0)
56     hgx = hgtransform;
57     set(hgx,'Matrix', makehgtform('zrotate',GMST0));
58     set(globe,'Parent',hgx);
59 end
60
61 %% Texturemap the globe
62
63 % Load Earth image for texture map
64
65 cdata = imread(image_file);
66
67 % Set image as color data (cdata) property, and set face
       color to indicate
68 % a texturemap, which Matlab expects to be in cdata. Turn off
       the mesh edges.
69
70 set(globe, 'FaceColor', 'texturemap', 'CData', cdata, '
       FaceAlpha', alpha, 'EdgeColor', 'none');
71
72 set(gca,'Clipping','off');
73
74 plot3(out.Posicion(:,1),out.Posicion(:,2),out.Posicion(:,3))
```

5.0.5. Datasheet Componentes

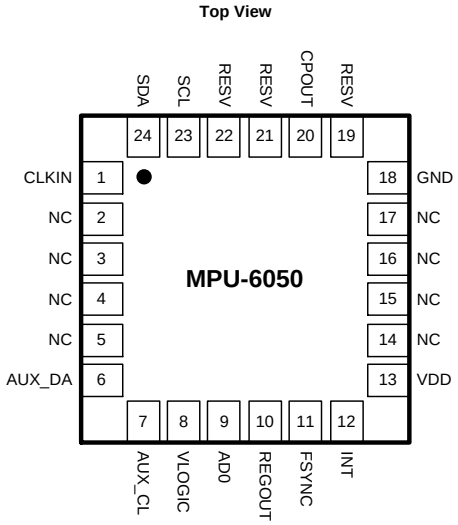
7 Applications Information

7.1 Pin Out and Signal Description

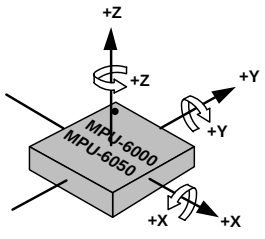
Pin Number	MPU-6000	MPU-6050	Pin Name	Pin Description
1	Y	Y	CLKIN	Optional external reference clock input. Connect to GND if unused.
6	Y	Y	AUX_DA	I^2C master serial data, for connecting to external sensors
7	Y	Y	AUX_CL	I^2C Master serial clock, for connecting to external sensors
8	Y		/CS	SPI chip select (0=SPI mode)
8		Y	VLOGIC	Digital I/O supply voltage
9	Y		AD0 / SDO	I^2C Slave Address LSB (AD0); SPI serial data output (SDO)
9		Y	AD0	I^2C Slave Address LSB (AD0)
10	Y	Y	REGOUT	Regulator filter capacitor connection
11	Y	Y	FSYNC	Frame synchronization digital input. Connect to GND if unused.
12	Y	Y	INT	Interrupt digital output (totem pole or open-drain)
13	Y	Y	VDD	Power supply voltage and Digital I/O supply voltage
18	Y	Y	GND	Power supply ground
19, 21	Y	Y	RESV	Reserved. Do not connect.
20	Y	Y	CPOUT	Charge pump capacitor connection
22	Y	Y	RESV	Reserved. Do not connect.
23	Y		SCL / SCLK	I^2C serial clock (SCL); SPI serial clock (SCLK)
23		Y	SCL	I^2C serial clock (SCL)
24	Y		SDA / SDI	I^2C serial data (SDA); SPI serial data input (SDI)
24		Y	SDA	I^2C serial data (SDA)
2, 3, 4, 5, 14, 15, 16, 17	Y	Y	NC	Not internally connected. May be used for PCB trace routing.



QFN Package
24-pin, 4mm x 4mm x 0.9mm

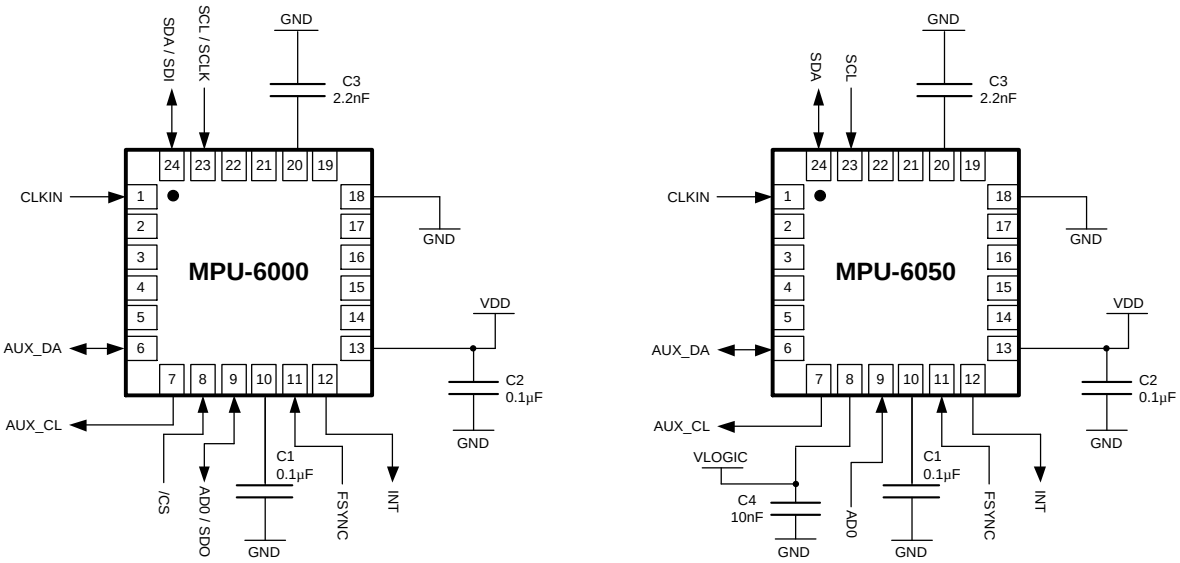


QFN Package
24-pin, 4mm x 4mm x 0.9mm



Orientation of Axes of Sensitivity and Polarity of Rotation

7.2 Typical Operating Circuit



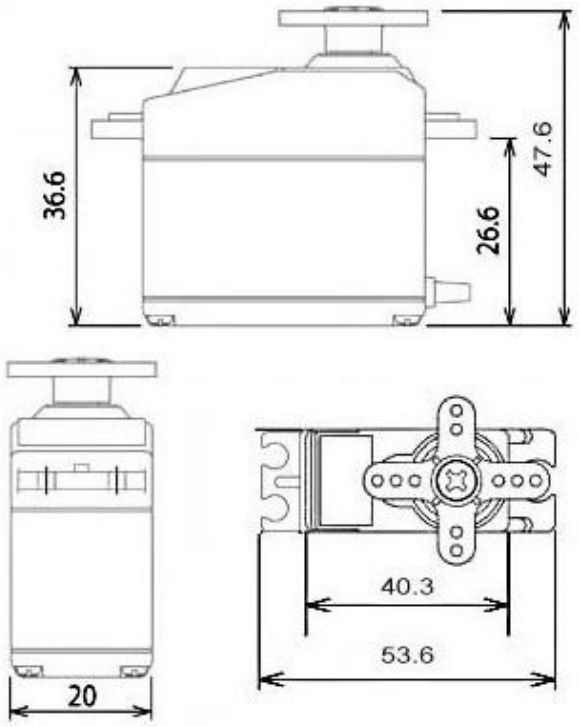
Typical Operating Circuits

7.3 Bill of Materials for External Components

Component	Label	Specification	Quantity
Regulator Filter Capacitor (Pin 10)	C1	Ceramic, X7R, 0.1µF ±10%, 2V	1
VDD Bypass Capacitor (Pin 13)	C2	Ceramic, X7R, 0.1µF ±10%, 4V	1
Charge Pump Capacitor (Pin 20)	C3	Ceramic, X7R, 2.2nF ±10%, 50V	1
VLOGIC Bypass Capacitor (Pin 8)	C4*	Ceramic, X7R, 10nF ±10%, 4V	1

* MPU-6050 Only.

MG996R High Torque
Metal Gear Dual Ball Bearing Servo



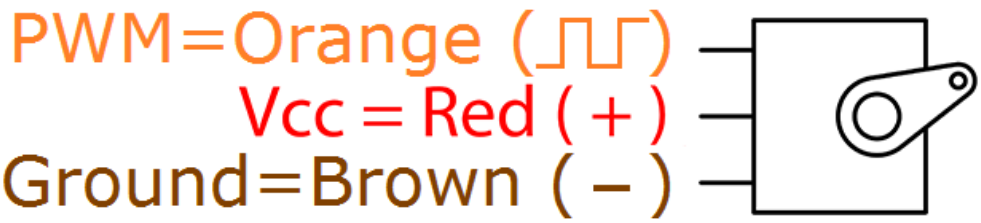
This High-Torque MG996R Digital Servo features metal gearing resulting in extra high 10kg stalling torque in a tiny package. The MG996R is essentially an upgraded version of the famous MG995 servo, and features upgraded shock-proofing and a redesigned PCB and IC control system that make it much more accurate than its predecessor. The gearing and motor have also been upgraded to improve dead bandwith and centering. The unit comes complete with 30cm wire and 3 pin 'S' type female header connector that fits most receivers, including Futaba, JR, GWS, Cirrus, Blue Bird, Blue Arrow, Corona, Berg, Spektrum and Hitec.

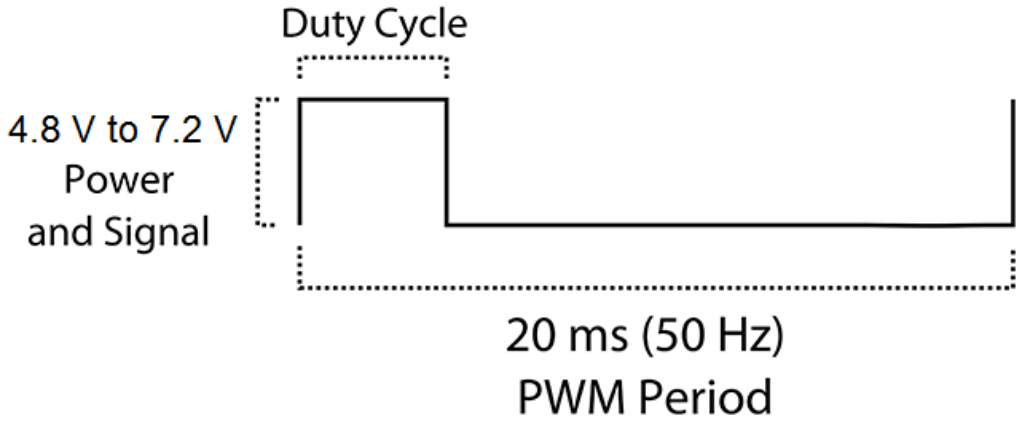
This high-torque standard servo can rotate approximately 120 degrees (60 in each direction). You can use any servo code, hardware or library to control these servos, so it's great for beginners who want to make stuff move without building a motor controller with feedback & gear box, especially since it will fit in small places. The MG996R Metal Gear Servo also comes with a selection of arms and hardware to get you set up nice and fast!

Specifications

- Weight: 55 g
- Dimension: 40.7 x 19.7 x 42.9 mm approx.
- Stall torque: 9.4 kgf·cm (4.8 V), 11 kgf·cm (6 V)
- Operating speed: 0.17 s/60º (4.8 V), 0.14 s/60º (6 V)

- Operating voltage: 4.8 V a 7.2 V
- Running Current 500 mA – 900 mA (6V)
- Stall Current 2.5 A (6V)
- Dead band width: 5 µs
- Stable and shock proof double ball bearing design
- Temperature range: 0 ºC – 55 ºC





Bibliografía

[1] Baesaudiovisual. *El Rey de los Lanzadores de Satélites Pequeños*. URL: https://lanzamientosespaciales.com/lanzadores-satelites-pequenos-compara/.

[2] Kirlin C. *An Introduction to Hardware-In-The-Loop (HIL) Simulation and its Applications*. International journal of advances in engineering, 2016.

[3] ARDUINO CC. *Arduino Mega 2560*. URL: https://store.arduino.cc/products/arduino-mega-2560-rev3.

[4] Artemio Josue Aguilar Cuellar. *Analisis y Simulacion de Trayectorias de Cohetes Meteorologicos*. Instituto Tecnologico y Estudios Superiores de Monterry, 2000.

[5] Rodriguez D. *Simulación de una Computadora de Vuelo para el Control de un Cohete Sonda*. Fundación Universitaria Los Libertadores, 2014.

[6] Jian Huang. «Design of Angle Detection System Based on MPU6050». En: *Proceedings of the 7th International Conference on Education, Management, Information and Computer Science (ICEMC 2017)* (2016).

[7] A. J. C. Godoy I. G. Perez y M. C. Godoy. *Fuzzy controller based on plc s7-´1200: Application to a servomotor*. International Conference on Informatics in Control, Automation y Robotics (ICINCO), 2014.

[8] H. Y. Irwanto. «HILS of auto Take Off System: For High Speed UAV Using Rooster Rocket». En: *International Seminar on Intelligent Technology and Its Applications (ISITIA)* (2016).

[9] MathWorks. *Hardware in the Loop (HIL)*. URL: https://www.mathworks.com/discovery/hardware-in-the-loop-hil.html.

[10] The MathWorks. *Getting Started with Arduino Hardware*. URL: https://www.mathworks.com/help/supportpkg/arduino/ref/getting-started-with-arduino-hardware.html.

[11] Nasa. *Flight of an Air Rocket*. URL: https://www.grc.nasa.gov/www/k-12/rocket/rktsflght.html.

[12] Nasa. *Flight to orbit*. URL: https://www.grc.nasa.gov/www/k-12/rocket/rktrflght.html.

[13] Katsuhiko Ogata. *Ingenieria de Control Moderna 5ª ED*. Prentice-Hall, 2010.

[14] OpenStax. *University Physics Volume 1*. CC BY (Attribution), 2016.

[15] Rincón Parra y Urrego. «Sistema de Control Simulado para Mantener la Trayectoria de un Cohete Balístico Atmosférico No-Teledirigido». En: *Desarrollo e innovación en ingeniería* (2016).

[16] K. Passino y S. Yurkovich. *Fuzzy Control, A.-W. Longman*. Ed. Columbus, Ohio, 1997.

[17] J. O. Murcia Piñeros. «Estudio de la trayectoria de un cohete de tres etapas lanzado desde el territorio colombiano». En: *Universidad Nacional* (2012).

[18] Jesus M. Recuenco. *Manual de Constructor de Modelos Espaciales Nivel de Iniciacion*. Creative Commons, 2008.

[19] RocketLab. «Payload user's guide». En: *Rocket Lab USA* (2022).

[20] SpaceX. *Halcon 9 Primer Cohete de Clase Orbital Capaz de Volver a Volar*. URL: `https://www.spacex.com/vehicles/falcon-9/`.

[21] Wallpaperbetterlogo. *Surtido de Cohetes Espaciales Ilustración, NASA, Espacio, Cohete, Infografía*. URL: `https://www.wallpaperbetter.com/es/hd-wallpaper-tcryl`.

[22] Huang Zhu & Zhao. «A Hardware-In-The-Loop Simulation Method for the Evaluation of Flight Control Systems». En: *International Conference on Automation, Control and Robotics Engineering (CACRE)* (2020).