INGENIERÍA AEROESPACIAL | PROYECTO PERSONAL

IMPLEMENTACIÓN Y SIMULACIÓN DE SISTEMAS GNC EN UAV

Desarrollo de plataforma experimental de cuadricóptero en Simulink para aprendizaje técnico

Autor: Manuel Ferraris | Febrero 2026 (v1.0)

github.com/1ManuFerraris/Personal-Project-UAV-SIMULATION

1. Introducción

Este proyecto consiste en la implementación de un sistema de control de vuelo para un vehículo aéreo no tripulado (tipo cuadricóptero), desarrollado y simulado íntegramente en el entorno **MATLAB/Simulink**. El trabajo abarca desde la dinámica del vehículo hasta la lógica de los lazos de control.

La elección de un cuadricóptero como plataforma de estudio responde a una decisión estratégica de **viabilidad**: a diferencia de sistemas más complejos o costosos (como satélites o aeronaves de ala fija de gran escala), los drones ofrecen una ruta accesible para una futura implementación en hardware físico y validación experimental, sin sacrificar la complejidad dinámica necesaria para el estudio de algoritmos de control robustos.

El objetivo principal es adquirir dominio sobre el software y familiarizarse con los conceptos fundamentales del área de Guiado, Navegación y Control (GNC). La motivación es generar una base práctica sólida **previa a la cursada formal de la materia**, lo que permitirá abordar posteriormente la teoría académica con un mejor entendimiento y mayor profundidad.

Dado que no se trata de un proyecto de diseño teórico, sino de **implementación de arquitectura y modelos**, se priorizó la velocidad de desarrollo. Para ello, se utilizaron modelos de Inteligencia Artificial como asistencia técnica para la generación de código, resolución de problemas y redacción del presente informe. Esta metodología permitió maximizar el alcance del proyecto, logrando abordar una mayor complejidad y contenido en un tiempo reducido.

2. Alcance del Proyecto

2.1 Modelo de Simulink

El desarrollo del trabajo abarcó la implementación progresiva de los siguientes 10 módulos y funcionalidades:

1. **Visualización en Tiempo Real:** Vinculación del entorno de simulación con **Unreal Engine** para la representación gráfica 3D de la física del vuelo.
2. **Entradas Human-in-the-Loop (HIL):** Integración de hardware de control (joystick físico) para comandar las entradas del sistema manualmente dentro del bucle de simulación.
3. **Dinámica de Cuerpo Rígido (6DOF):** Modelado matemático de las ecuaciones de movimiento en dos instancias: una **implementación manual** de las ecuaciones para comprensión teórica, y una implementación final optimizada utilizando el bloque *6DOF* del **Aerospace Blockset**.

4. **Sistema de Propulsión y Aerodinámica:** Implementación de modelos de actuadores (motores/hélices), matriz de mezcla (Mixer) y fuerzas aerodinámicas básicas.

5. **Arquitectura de Control:** Diseño y sintonización de lazos PID en cascada, permitiendo la conmutación en vuelo entre modos **Acro, Stabilizer y GPS**.

6. **Estimación de Estados y Ruido:** Inyección de ruido en sensores (IMU, GPS, Barómetro, Magnetómetro) e implementación de filtros para la **estimación de la actitud y posición** del vehículo.

7. **Paneles de Interfaz y Hardware Virtual:** Desarrollo de dos tableros de operación: un panel de **Telemetría** y un panel de **Hardware** (simulación de encendido, voltaje de batería y LEDs de estado).

8. **Perturbaciones Ambientales:** Incorporación de modelos de viento y turbulencia (Dryden), configurables desde un panel externo para evaluar la respuesta ante disturbios.

9. **Arquitectura de Software de Vuelo:** Aplicación de metodologías de *Model-Based Design* y configuración de una ejecución *Multi-rate* (frecuencias diferenciadas para lazos rápidos de control y lentos de navegación).

10. **Validación Integral (SIL):** Verificación final del sistema mediante simulación *Software-in-the-Loop*, asegurando la portabilidad y robustez del código generado.

2.2 Escenario de Validación Funcional

Como etapa final, se desarrolló una simulación de misión para observar el comportamiento integrado de los subsistemas en un entorno operativo. El objetivo es describir la operación de los controladores y la lógica de modos bajo condiciones dinámicas, ilustrando las transiciones entre Acro, Stabilizer y GPS. Asimismo, se muestra el funcionamiento de los lazos de control frente a ráfagas de viento y turbulencia para observar la capacidad de compensación del sistema.

A continuación, se detalla la secuencia de la misión:

0. Secuencia de iniciación y despegue: Tras la conexión de batería, encendido y calibración de sensores, se realiza una prueba de seguridad donde el armado se rechaza por aceleración elevada (*throttle*). Una vez corregido el mando, se procede al armado y despegue vertical.

1. Prueba de control en modo Acro: Verificación de la respuesta directa de los lazos de velocidad angular.

2. Prueba de control en modo Stab: Operación de los controladores de actitud para mantener el nivel del vehículo.

3. Prueba de control en modo GPS: Activación del control de velocidad para el mantenimiento de posición y estabilización de la velocidad vertical.

4. Respuesta a perturbaciones atmosféricas: Inyección de ráfagas de viento y turbulencia para observar el comportamiento de los controladores.

5. Recorrido en modo Acro: Navegación manual sin asistencia de nivelación.

6. Acrobacia (Acro Roll): Ejecución de una maniobra de rotación completa de 360° sobre el eje longitudinal.

7. Reorientación del estimador de actitud: Comportamiento de los filtros para recuperar la referencia de nivel tras la maniobra dinámica.

8. Recorrido en modo Stab: Navegación asistida por el control de actitud.

9. Recorrido en modo GPS: Control de velocidad constante para el desplazamiento guiado por el operador.

10. Aterrizaje: Maniobra final de descenso y desactivación del sistema.

Fig. 2.2. *Trayectoria y secuencia de validación funcional.*

2.3 Exportación y Análisis de Datos

Para facilitar el estudio de la dinámica del vehículo, se desarrolló un script que permite exportar la telemetría completa de la simulación en archivos de formato **.csv**. Esta funcionalidad permite el procesamiento de datos en herramientas externas para evaluar el comportamiento de los lazos de control frente a las maniobras ejecutadas. Como parte de este trabajo, se incluye la base de datos obtenida durante las simulaciones realizadas, la cual sirvió de sustento para la validación de la arquitectura implementada. Además, se han incorporado en el **Anexo A** las gráficas de mayor relevancia que fueron utilizadas como referencia constante durante las etapas de construcción y ajuste del proyecto, permitiendo visualizar el proceso de sintonización de los filtros de estimación.

3. Arquitectura General del Sistema

El diseño de la simulación se rige estrictamente por los principios del **Diseño Basado en Modelos (Model-Based Design)**. Bajo este paradigma, la arquitectura de nivel superior no es arbitraria, sino que desacopla explícitamente el entorno de operación, el software embebido y la física del vehículo para facilitar la validación y la futura generación de código.

El modelo (*Top-Level Model*) segrega el sistema en tres subsistemas independientes que interactúan en un lazo cerrado:

1. **Pilot Interface (Las Entradas):** Abstrae la interacción humana, inyectando los comandos externos (joystick) necesarios para excitar el sistema y visualizando la telemetría de retorno.
2. **Flight Controller Software (El Código):** Representa el algoritmo de control discreto. En un esquema MBD, este bloque es el único que contiene la lógica que eventualmente sería implementada en un microcontrolador, estando aislado de la física para permitir verificaciones tipo *Software-in-the-Loop (SIL)*.
3. **Vehicle Interface (La Planta):** Representa la realidad física (dinámica continua, actuadores y sensores). Su función es cerrar el lazo de retroalimentación, proveyendo al controlador los estímulos que recibiría en un vuelo real.

Esta separación modular asegura que el controlador no "sepa" que está en una simulación, operando únicamente a través de interfaces de señales estandarizadas.

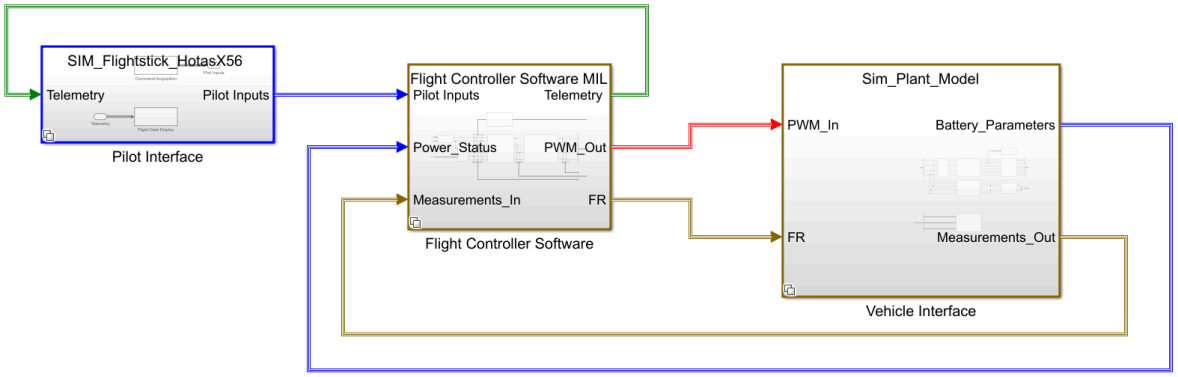

Fig.3. *Arquitectura de Nivel Superior (Top-Level Model) del entorno de simulación.*

Dado que el sistema opera bajo una arquitectura **Multi-rate** para emular la ejecución en tiempo real de un procesador embebido, se utiliza el siguiente código de colores para identificar la frecuencia de muestreo (*Sample Time*) de cada bus de señales:

- **Rojo:** 1000 Hz (1 ms)
- **Amarillo:** 250 Hz (4 ms)
- **Azul:** 100 Hz (10 ms)
- **Verde:** 50 Hz (20 ms)

A continuación, se describen en detalle la función y composición interna de cada uno de estos bloques.

3.1. Bloque Pilot Interface

Este módulo constituye la interfaz de comando del vehículo, agrupando todas las interacciones remotas del piloto y simulando el comportamiento de un sistema de radiocontrol. Gracias a su arquitectura de *Variant Subsystem*, funciona como una capa de abstracción: se puede alternar entre drivers de hardware físico (como un Joystick USB) o futuros drivers de radiocontrol real, sin alterar la lógica interna del controlador de vuelo. Esto permite incorporar dispositivos externos para cerrar el lazo de control con un operador real ("Human-in-the-Loop"). Internamente posee dos subsistemas:

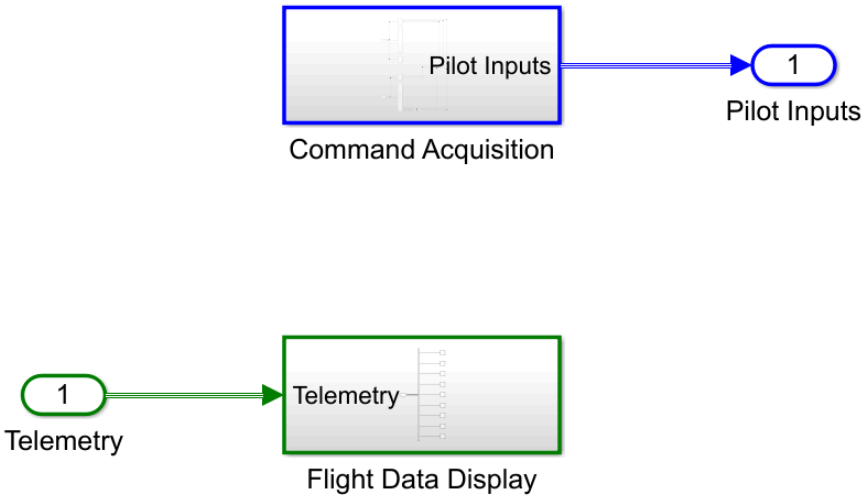

Fig. 3.1. *Arquitectura interna del bloque Pilot Interface y sus submódulos.*

3.1.1 Command Acquisition

Este subsistema gestiona la adquisición de datos en tiempo real provenientes del hardware físico (Joystick Logitech X 56). Su función principal es el acondicionamiento de señal: se encarga de identificar, mapear y normalizar las entradas crudas del joystick para hacerlas compatibles con el controlador. Opera a una frecuencia de 100 Hz, replicando la tasa de actualización estándar de los enlaces de radiocontrol de baja latencia.

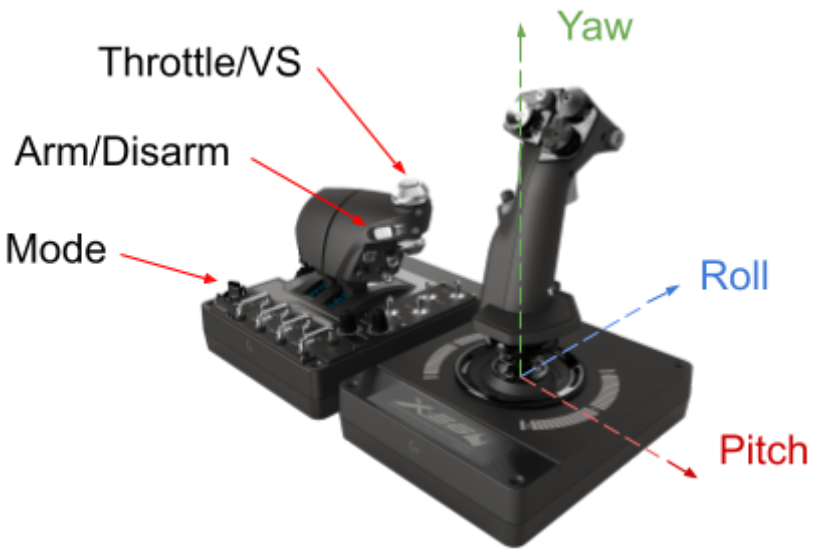

Fig. 3.1.1-A Hardware Logitech X56 utilizado para las pruebas *Human-in-the-Loop*.

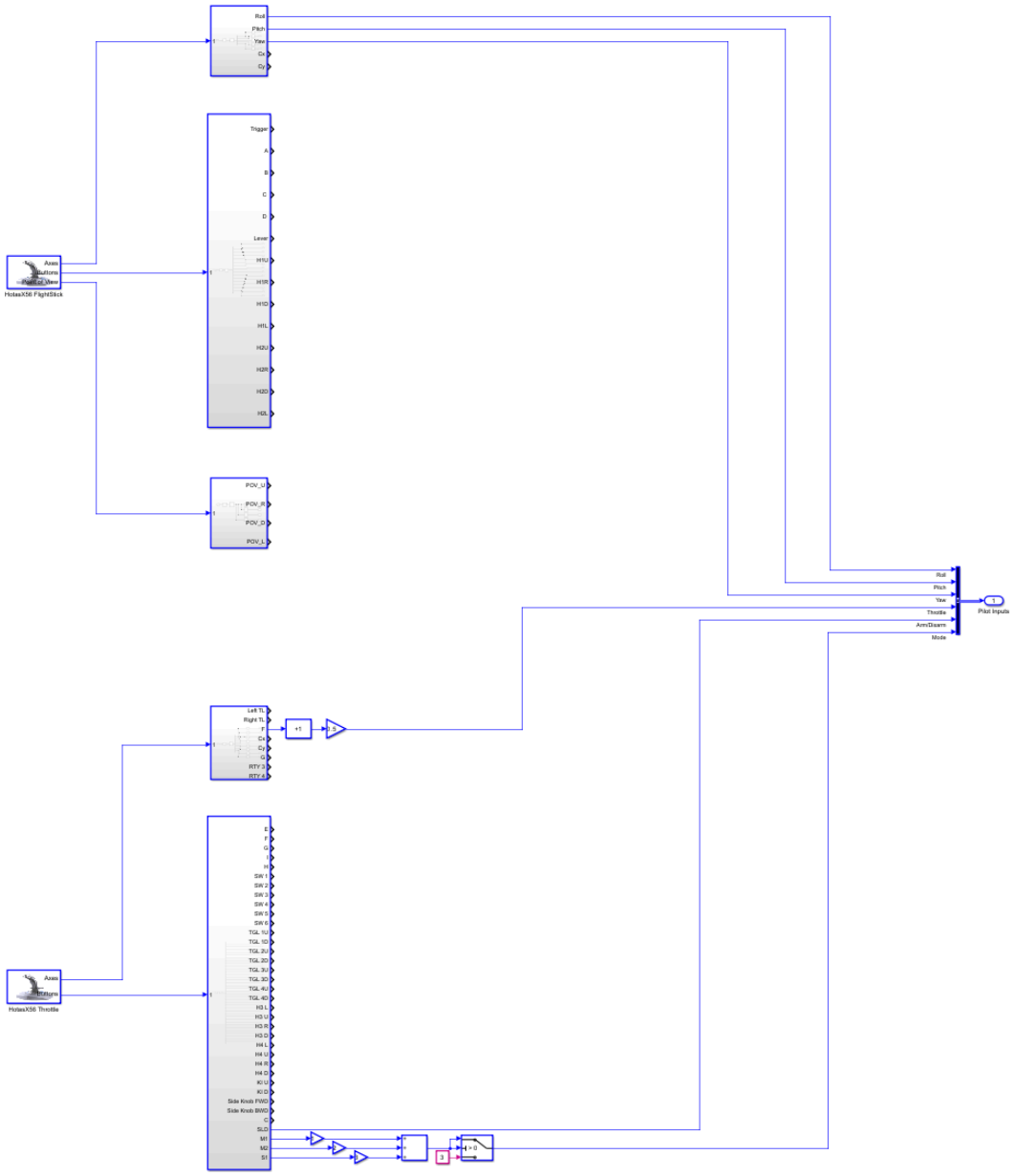

Fig. 3.1.1-B *Lógica interna de acondicionamiento y normalización de señales de entrada.*

3.1.2 Flight Data Display

Su finalidad es la visualización de telemetría, vinculando las variables de estado del dron con un panel de instrumentos virtual. Este módulo opera a una frecuencia reducida de **50 Hz**; esta decisión de diseño (submuestreo) busca simular las restricciones reales de ancho de banda en los protocolos de transmisión de datos y optimizar el rendimiento computacional de la simulación.

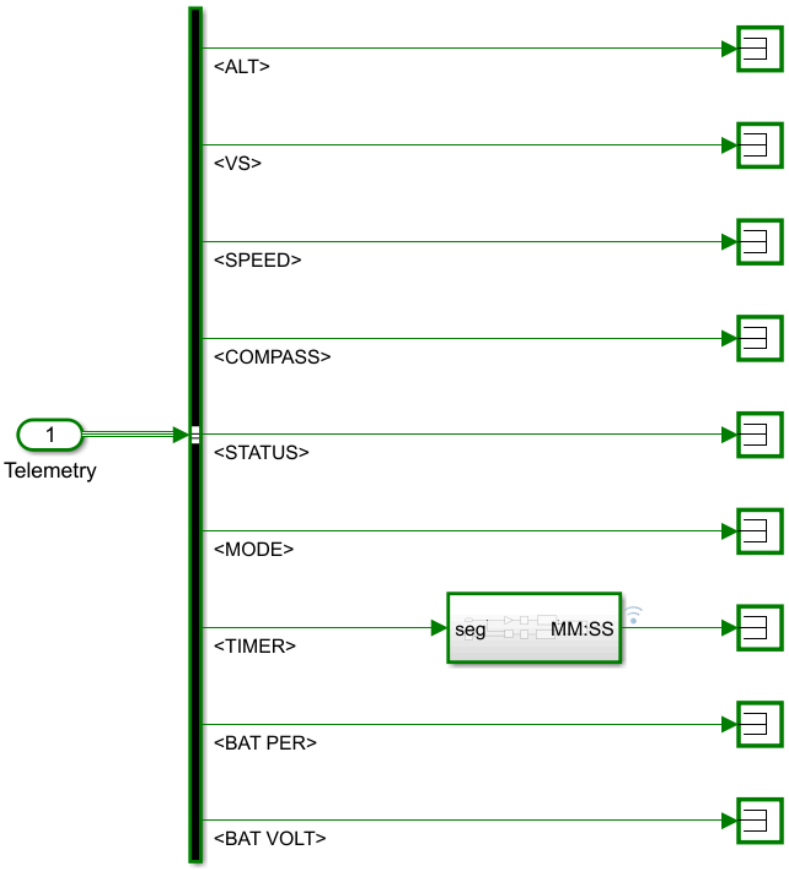

Fig. 3.1.2 *Conexión interna del bus de telemetría con los elementos visuales del Dashboard.*

3.2. Bloque Flight Controller Software

Este bloque constituye el núcleo computacional o "firmware" del vehículo. Operando estrictamente en el dominio del tiempo discreto, simula la ejecución cíclica de un microcontrolador real, procesando entradas y generando comandos de control a una frecuencia fija.

Una característica distintiva de su arquitectura es su implementación como Variant Subsystem. Esto otorga la flexibilidad de conmutar instantáneamente entre dos modos de ejecución:

1. **MIL (Model-in-the-Loop):** Ejecución interpretada para iteración rápida y depuración de algoritmos.
2. **SIL (Software-in-the-Loop):** Ejecución de código C/C++ generado automáticamente y compilado, permitiendo validar la fidelidad y tiempos de ejecución del código embebido final frente al modelo matemático.

Internamente, se organiza en cinco subsistemas funcionales:

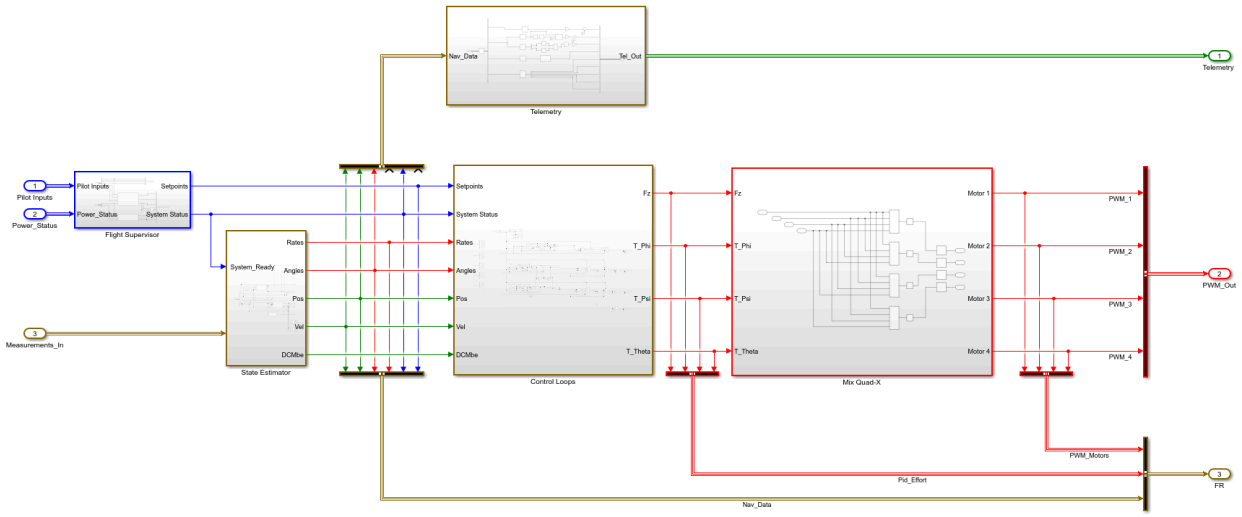

Fig. 3.2. *Arquitectura interna del subsistema Flight Controller Software.*

3.2.1. Flight Supervisor

Este módulo actúa como el gestor lógico de alto nivel del sistema. Su función principal es centralizar la toma de decisiones discretas y procesar las variables operativas críticas para alimentar al sistema de telemetría.

Internamente, el bloque desempeña dos roles fundamentales:

1. Máquina de Estados de Seguridad (FSM): Implementada sobre Stateflow, esta sección define una estricta lógica secuencial que gobierna el ciclo de vida del vuelo y la seguridad operativa:

- Secuencia de Inicialización (Init): Al energizar el sistema (Power On), la FSM entra en el estado Init. Durante una ventana fija de 3 segundos, el sistema bloquea cualquier comando de control para ejecutar la calibración de bias de los sensores inerciales.
- Estado Disarmed (Standby): Finalizada la calibración, el sistema transiciona automáticamente a Disarmed (motores desactivados), quedando a la espera del comando del piloto.
- Estado Armed (Active): Desde *Disarmed*, si el piloto acciona el interruptor de armado y se cumplen todas las condiciones de seguridad, el sistema transiciona a Armed. En este estado, se habilitan las salidas PWM hacia los motores, se activan los lazos de control y comienza a correr el cronómetro de misión (*Flight Timer*).
- Gestión de Errores y Enclavamientos (State Rejected): Se implementaron condiciones de rechazo (Interlocks) para prevenir arranques accidentales:

– Pre-Arm Check: Si el interruptor de armado está activo *durante* la fase de inicialización.

– Throttle Check: Si se intenta armar el vehículo con el acelerador (*Throttle*) por encima del mínimo.

- En cualquiera de estos casos, la máquina deriva al estado de fallo Rejected. Para recuperar el sistema, se obliga al piloto a realizar una maniobra de seguridad: debe devolver el interruptor de armado a la posición OFF (reset) para regresar al estado *Disarmed* y reintentar la secuencia bajo condiciones seguras.

2. Procesamiento de Datos y Telemetría: Paralelamente, el subsistema acondiciona la información del estado del vehículo para el piloto:

- Estimación de Autonomía (SoC): Convierte la lectura de tensión del ADC en un porcentaje de Estado de Carga (0-100%) mediante una linealización, permitiendo monitorear la energía remanente.
- Cronometría de Misión: Gestiona un *Flight Timer* que contabiliza el tiempo efectivo de vuelo (mientras el estado es *Armed*).
- Codificación de Modos: Estandariza la señal del modo de vuelo activo (Acro, Stabilize, GPS) para su transmisión al Dashboard.

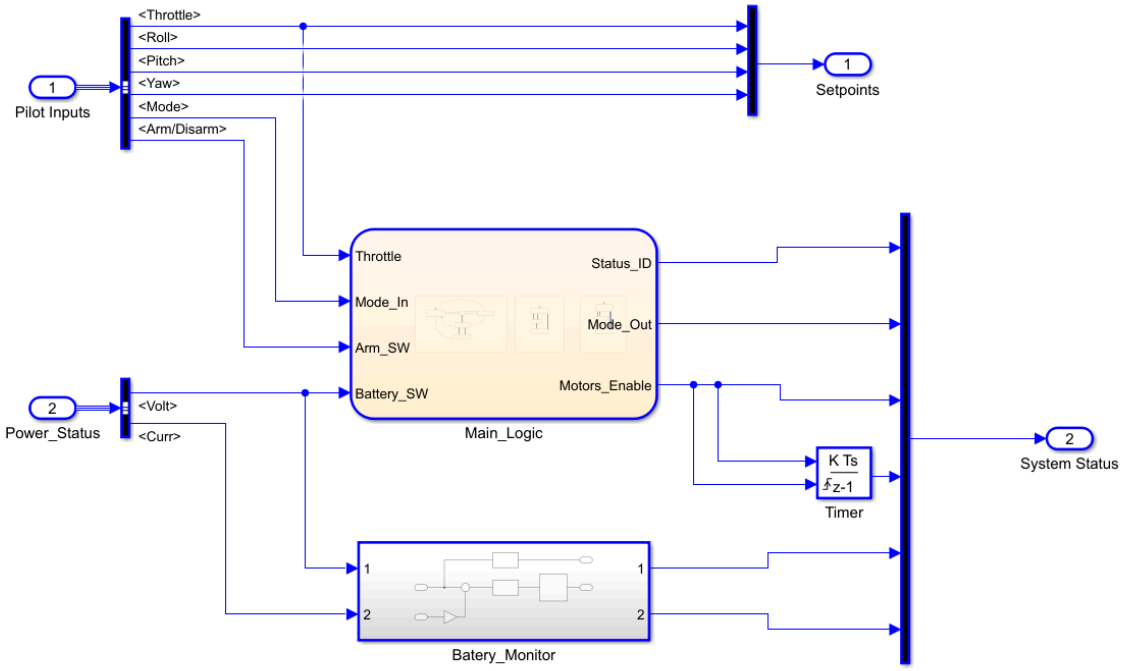

Fig. 3.2.1-A *Arquitectura interna del Flight Supervisor*

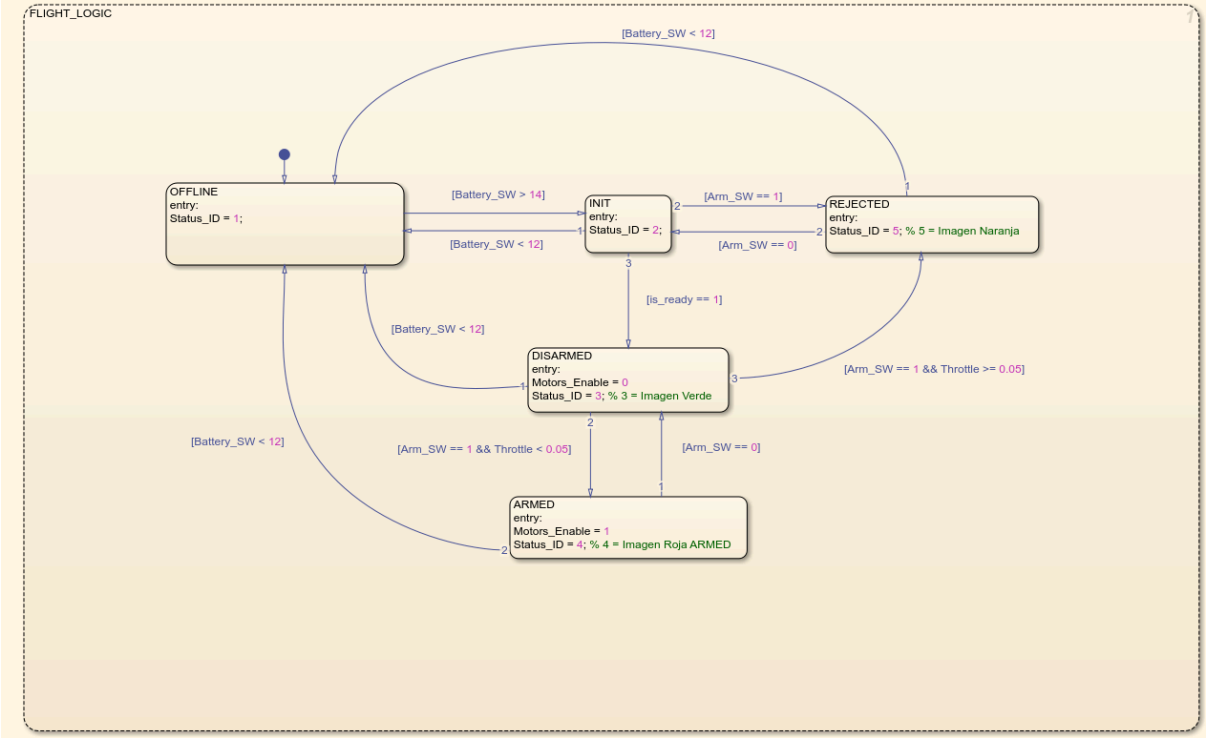

Fig. 3.2.1-B *Lógica de Vuelo Principal y Seguridad de Armado.*

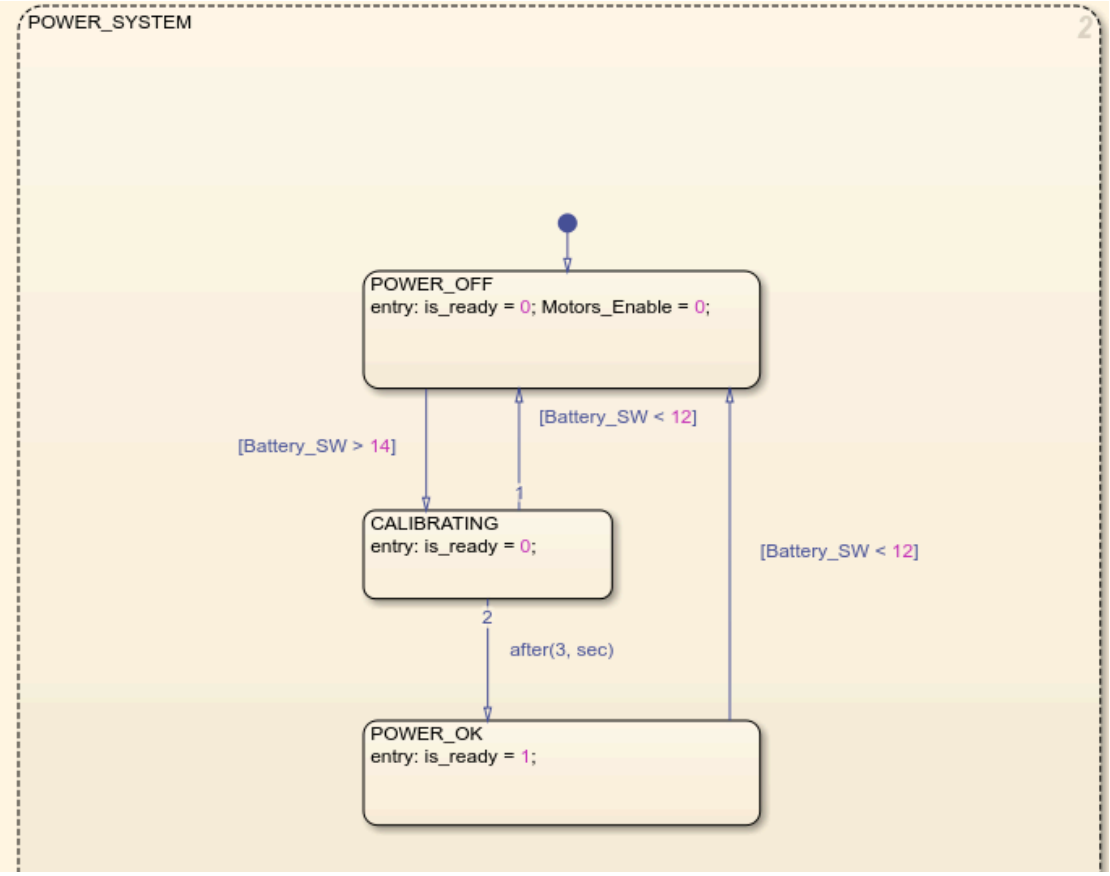

Fig. 3.2.1-C *Lógica de Gestión de Energía y Secuencia de Calibración de Sensores.*

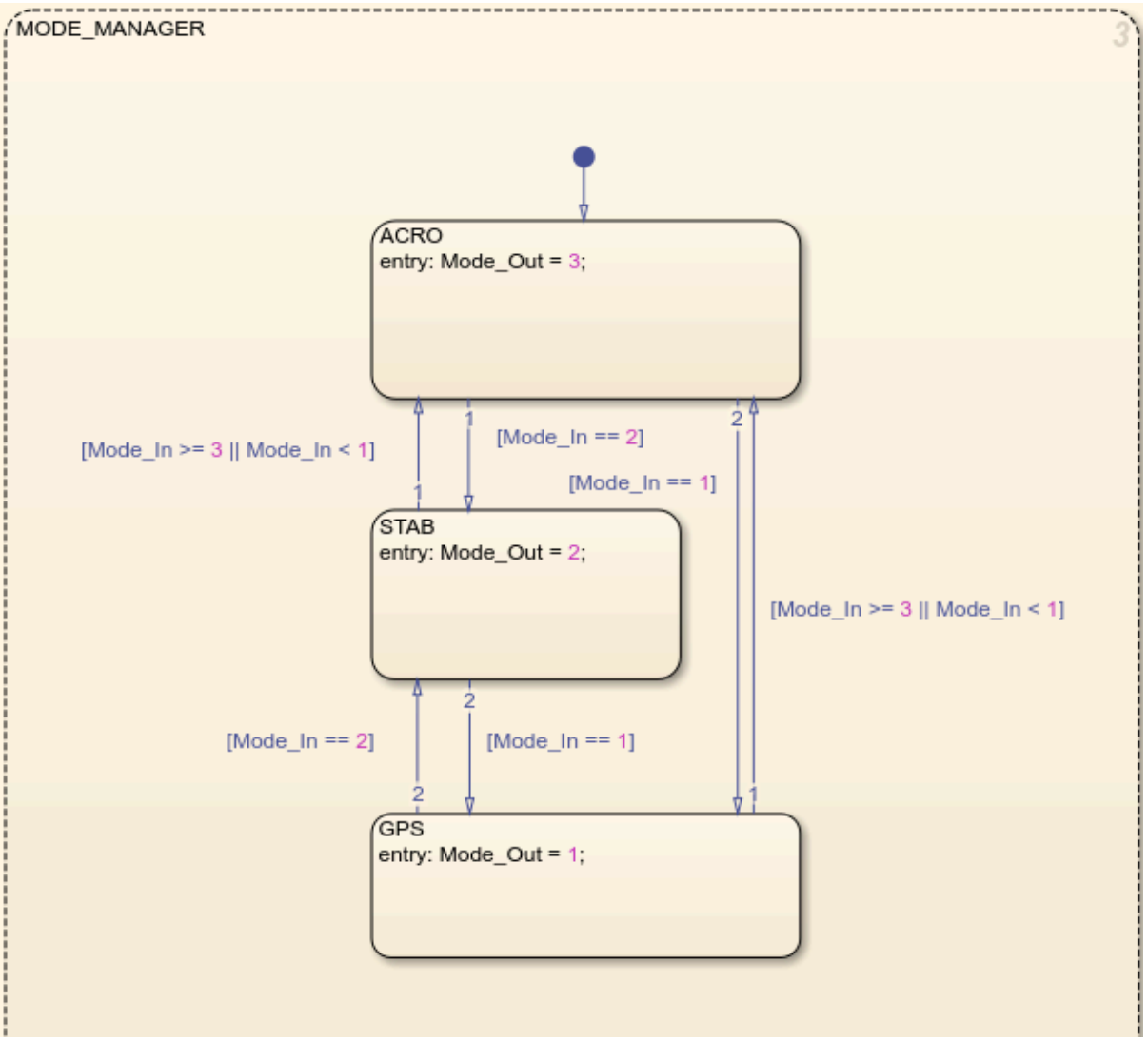

Fig. 3.2.1-D *Lógica de Selección y Conmutación de Modos de Vuelo.*

3.2.2. State Estimator

El subsistema de Estimación de Estado constituye el núcleo de navegación del vehículo. Su función es procesar las mediciones de los sensores a bordo (IMU, Magnetómetro, GNSS y Barómetro) para reconstruir el estado cinemático completo del dron. La arquitectura se basa en una topología en cascada de dos Filtros de Kalman Extendidos (EKF), lo que permite desacoplar las dinámicas rápidas de rotación de las dinámicas lentas de traslación.

Durante la fase de desarrollo y sintonización de los filtros, se implementó una arquitectura de validación dual mediante el uso de switches manuales en la etapa de retroalimentación. Esta configuración permitía conmutar dinámicamente la entrada de los lazos de control (PID) entre las variables de Ground Truth y las estimaciones generadas por los EKF.
Este esquema de pruebas fue fundamental por dos razones:

- **Aislamiento de Errores:** Permitió que los filtros actuaran exclusivamente como observadores pasivos mientras el dron mantenía un vuelo estable gobernado por el Ground Truth, facilitando la identificación de sesgos o derivas en los sensores sin riesgo de inestabilidad en la planta.
- **Calibración Comparativa:** Mediante el uso de *scopes* y bloques de registro conectados directamente a las señales de referencia ideal, se ajustaron las matrices de covarianza de ruido de proceso (Q) y de medición (R) hasta lograr una convergencia óptima.

Una vez validada la precisión y estabilidad de los estimadores, esta lógica de conmutación y los bloques de comparación directa fueron removidos de la arquitectura de vuelo, consolidando un sistema de navegación autónomo donde el control depende exclusivamente de los estados reconstruidos por el State Estimator.
Los scripts de implementación correspondientes a ambos algoritmos de fusión se encuentran detallados en el Anexo B (Códigos B.3 y B.4).

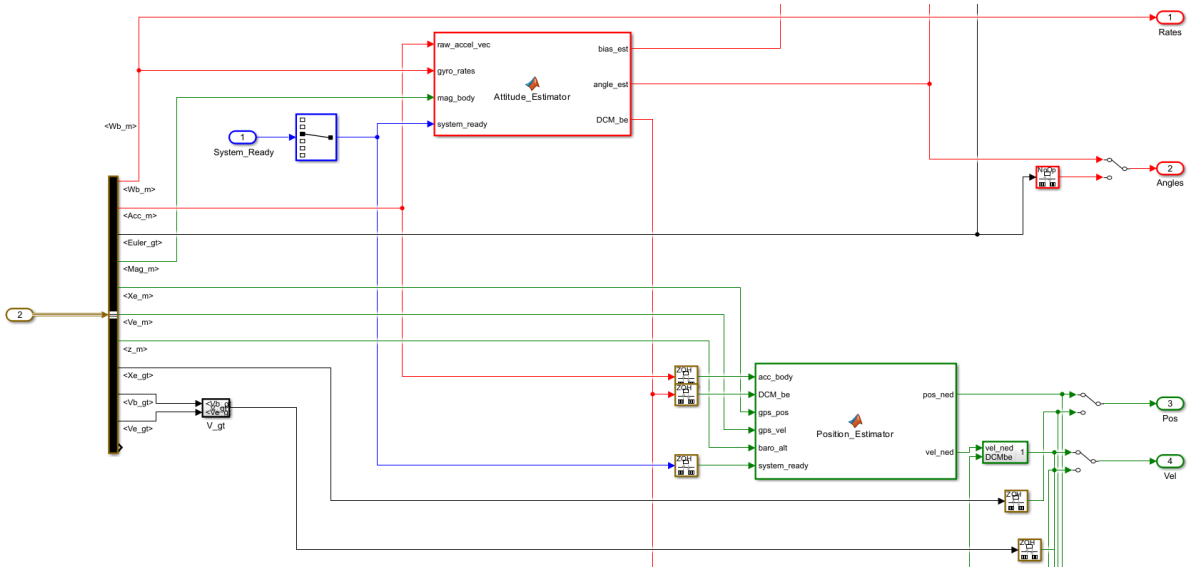

Fig. 3.2.2.-A *Configuración de prueba para calibración de filtros mediante comparación con Ground Truth.*

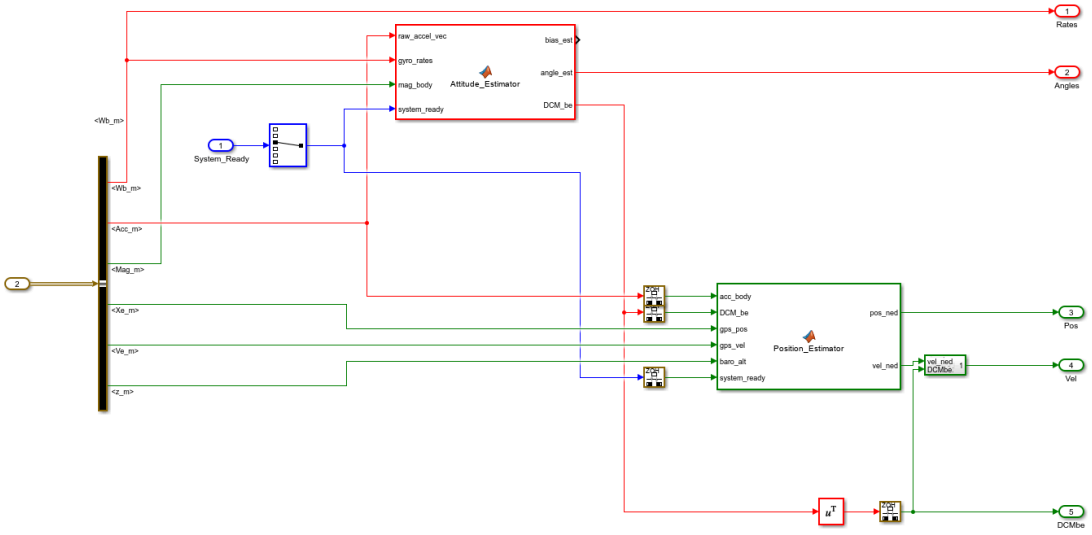

Fig. 3.2.2.-B *Arquitectura final del estimador de estados.*

1. Estimador de actitud: este bloque implementa un algoritmo de fusión sensorial ejecutado a una frecuencia de **1 kHz**. Su objetivo es estimar la orientación (Roll, Pitch, Yaw) y compensar los sesgos (*biases*) de los giroscopios en tiempo real. El vector de estado $x \in R^6$ se define como:

$$x = [\phi,\ b_p,\ \theta,\ b_q,\ \varphi,\ b_r\ ]$$

Donde ϕ,θ, φ representan los ángulos de Euler y b_p, b_q, b_r estiman la deriva dinámica de los giroscopios.

El funcionamiento del algoritmo se divide en las siguientes etapas:

- **Inicialización y Calibración:** Al encendido, el sistema ejecuta una rutina de 3 segundos para capturar el *bias* estático del giroscopio y determinar el *offset* magnético inicial, estableciendo el rumbo actual como referencia cero (*Heading Relativo*).
- **Etapa de Predicción:** Se integra la velocidad angular medida por el giroscopio (ω) utilizando las ecuaciones cinemáticas de Euler discretizadas. Simultáneamente, se propaga la matriz de covarianza del error P considerando el ruido del proceso Q
- **Corrección Gravitacional (Roll/Pitch):** Se utiliza el vector de aceleración para corregir la actitud. El algoritmo incluye una lógica de **Rechazo de Aceleraciones Dinámicas**: si la magnitud del vector de fuerza se desvía del umbral gravitatorio ($||a|| \approx 9.81 \pm 0.6\, m/s^2$), el filtro "congela" la corrección para evitar errores inducidos por fuerzas centrífugas o maniobras bruscas.
- **Corrección Magnética (Yaw):** Se emplea un magnetómetro triaxial con **Compensación de Inclinación (*Tilt Compensation*)**. Los datos magnéticos

son rotados al plano horizontal utilizando las estimaciones de Roll y Pitch previas, desacoplando así la estimación de rumbo de la inclinación del vehículo.

2. Estimador de posición: Este subsistema implementa un algoritmo de Navegación Inercial Asistida (INS/GPS) operando a una frecuencia de **50 Hz**. Su función principal es fusionar la alta frecuencia de actualización de la IMU con la precisión absoluta del GPS y el barómetro. El vector de estado $x_{pos} \in R^6$ representa la cinemática traslacional en el marco NED:

$x_{pos} = [P_n, P_e, P_d, V_n, V_e, V_d]^T$

Donde P_n, P_e, P_d son las coordenadas de posición relativa y V_n, V_e, V_d las velocidades lineales en los ejes Norte, Este y Abajo respectivamente.

La lógica de procesamiento sigue el siguiente esquema:

- **Inicialización y Calibración Estática:** En estado de reposo (*Ground Mode*), el sistema realiza un promedio de las lecturas barométricas para establecer el "Cero Local" (P_d=0) y ancla la posición horizontal a la primera solución válida del GNSS, eliminando *offsets* iniciales.
- **Mecanización Inercial (*Strapdown*):** Se aplica un filtrado paso bajo al acelerómetro para reducir ruido vibracional. Posteriormente, el vector de aceleración es rotado del marco del cuerpo al marco inercial utilizando la Matriz de Cosenos Directores (DCM) proveniente del Estimador de Actitud, compensando el vector gravedad ($g = 9.81 m/s^2$) para obtener la aceleración cinemática neta.
- **Etapa de Predicción:** Se propagan los estados mediante integración numérica de las leyes de Newton. El modelo dinámico incluye un coeficiente de **Arrastre Aerodinámico Lineal (*Drag Damping*)**, que actúa como un amortiguador matemático para evitar la divergencia de la velocidad ante derivas del acelerómetro.
- **Corrección Asíncrona (*Multi-Rate Fusion*):** El filtro maneja las distintas tasas de muestreo de los sensores mediante una lógica selectiva:

  – Canal Vertical: Se corrige continuamente utilizando la altitud barométrica inversa.

  – Canal Horizontal: Se detecta la llegada de nuevas tramas GPS (típicamente a 5-10 Hz). Solo cuando hay datos frescos, la matriz de observación H se expande dinámicamente para corregir posición y velocidad horizontal.

3.2.3. Control Loops

El subsistema de Control constituye la etapa de actuación del vehículo. Su arquitectura implementa una estrategia de **Control PID en Cascada (*Cascaded PID*)**, diseñada para gobernar las variables dinámicas del dron de manera jerárquica. El sistema procesa el vector de estado estimado y las referencias de mando para generar las señales de mezcla (PWM) que controlan la velocidad de los motores, asegurando la estabilidad y el seguimiento de trayectoria.

La topología del sistema se organiza en tres lazos de realimentación anidados con frecuencias de actualización decrecientes, los cuales se habilitan progresivamente mediante la señal de selección *Mode*:

- **Lazo Interno - Control de Tasa (*Rate Loop*):**
  Ejecutado a una frecuencia de **1000 Hz**, regula la dinámica rotacional rápida del vehículo (p,q,r). Este lazo compara la velocidad angular medida por el giroscopio con la referencia deseada y actúa directamente sobre el par motor. Permanece activo en todos los modos de vuelo, garantizando la estabilización base del sistema con la menor latencia posible.
- **Lazo Intermedio - Control de Actitud (*Attitude Loop*):**
  Operando a **250 Hz**, supervisa la orientación angular $(\phi,\theta,)$ del vehículo. Este controlador procesa el error de ángulo y su salida no se envía a los actuadores, sino que se convierte en la **referencia dinámica (*Setpoint*)** para el lazo interno de Tasa. Incluye saturaciones de seguridad para limitar la inclinación máxima permitida.
- **Lazo Externo - Control de Posición/Velocidad (*Position Loop*):**
  Nivel superior de automatización ejecutado a **50 Hz**, gestiona la posición y velocidad lineal en el marco de navegación (P_{ned}, V_{ned}). Este lazo genera los comandos de inclinación (Roll/Pitch) necesarios para anular el error de posición, cerrando el ciclo de navegación autónoma al inyectar estas referencias en el lazo de Actitud.

El sistema integra mecanismos de supervisión gobernados por la señal global System Ready y una lógica de selección de señales diseñada para mantener la estabilidad en las transiciones de modo:

- **Conmutación de Errores:** Se utiliza una lógica de switches que fuerza el error a cero en los controladores PID que no se encuentran operativos en el modo de vuelo actual. Esto anula la respuesta de los términos proporcionales y derivativos, permitiendo una transición limpia entre estados.

- **Seguridad de Armado:** La señal System Ready funciona como interruptor maestro de seguridad. En estado desarmado, se anula la transmisión de señal a los ESCs, impidiendo el giro accidental de los motores

independientemente de las entradas de control y garantizando la integridad del operador.

- **Prevención de Windup mediante Reset:** Se aplica un reinicio constante (Reset) a los controladores que no están en uso, ya sea por estar el sistema desarmado o por no pertenecer al modo de vuelo activo. Esta acción garantiza que los integradores inicien desde un estado nulo al activarse el lazo, eliminando saltos bruscos en la salida causados por errores acumulados mientras el control permanecía abierto.

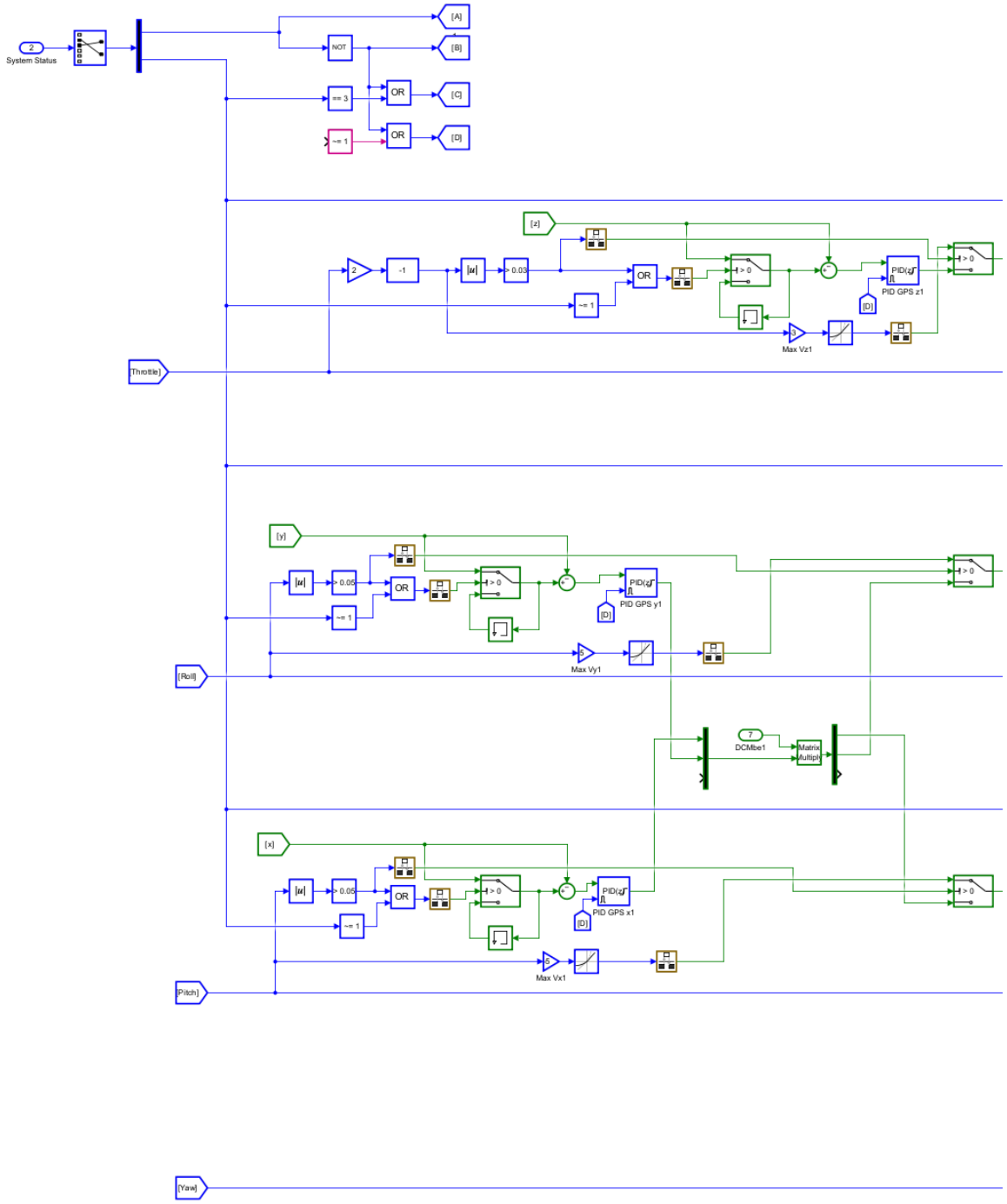

Fig. 3.2.3-A *Arquitectura de Control en Cascada: Lazos Exteriores de Navegación.*

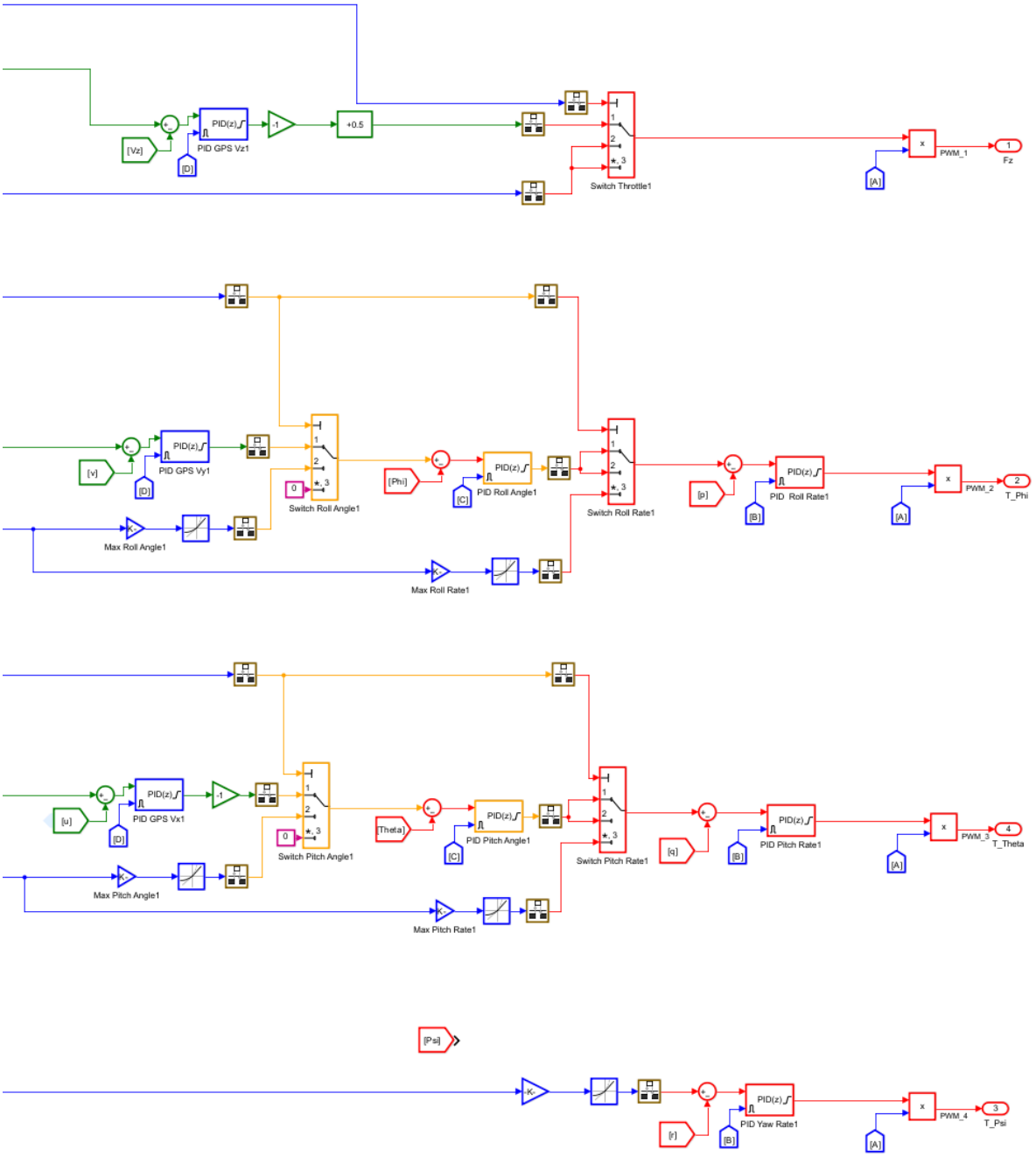

Fig. 3.2.3-B *Arquitectura de Control en Cascada: Lazos Interiores de Estabilización.*

3.2.4. Mix Quad-x

El bloque de Mezcla o *Mixer* constituye la interfaz final del sistema de navegación, encargada de realizar la **Asignación de Control (*Control Allocation*)**. Su función es transformar el vector de fuerzas y momentos generalizados $(F_z, T_\phi, T_\theta, T_\varphi)$, calculado por los lazos de control PID, en comandos individuales de velocidad angular para cada uno de los cuatro motores del cuadricóptero.

1. Distribución Geométrica: El algoritmo implementa una matriz de mezcla estática diseñada para una configuración de fuselaje en "X". Las entradas de control se combinan linealmente mediante sumas y restas algebraicas para resolver la contribución requerida de cada rotor. Esto permite que el vehículo ejecute maniobras acopladas, generando simultáneamente sustentación vertical (Thrust) y pares de rotación (Torque) en los tres ejes principales.

2. Linealización y Acondicionamiento de Señal: El subsistema incorpora etapas de procesamiento para adaptar la señal a la dinámica física de los actuadores:

- **Compensación de Empuje (*Thrust Linearization*):**
  Se aplica una función de **raíz cuadrada** ($\sqrt{u}$) a la salida de la mezcla. Dado que el empuje generado por una hélice es proporcional al cuadrado de su velocidad angular ($F \propto \omega^2$), esta operación linealiza la respuesta del sistema, permitiendo que los controladores PID operen bajo la asunción de un sistema lineal.
- **Saturación de Actuadores:**
  Cada canal de motor incluye bloques de saturación dinámica. Estos limitadores aseguran que las señales enviadas a los Variadores Electrónicos de Velocidad (ESC) se mantengan dentro del rango operativo físico (0% a 100% de potencia), protegiendo los motores de comandos inviables (valores negativos) o de sobrecargas.

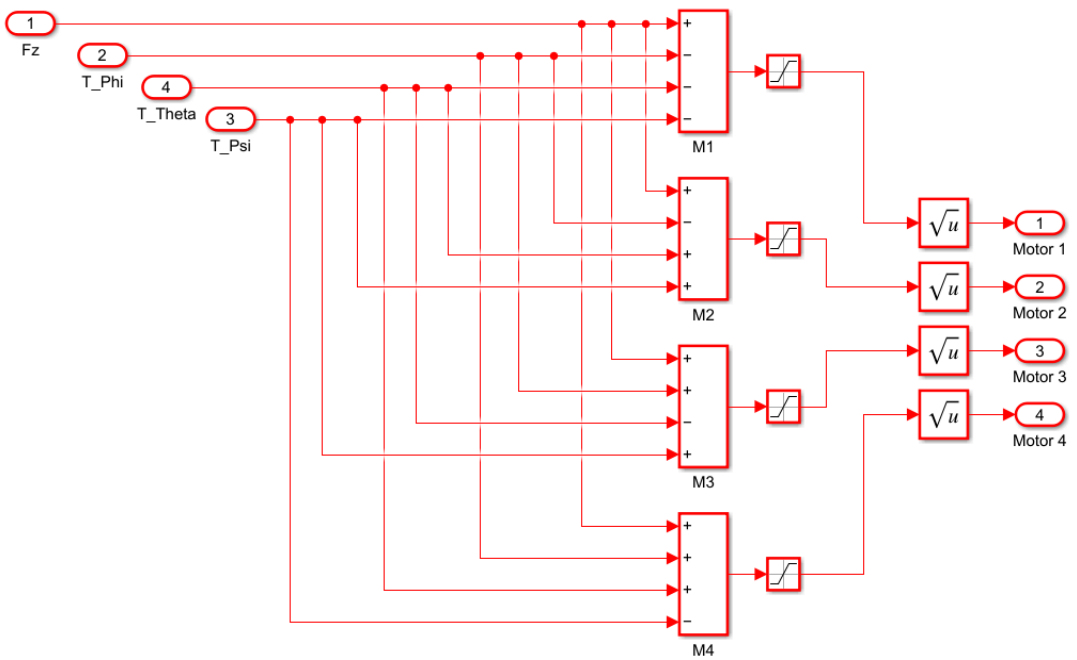

Fig. 3.2.4 *Implementación de la Matriz de Mezcla para configuración Quad-X.*

3.2.5. Telemetry

Este subsistema actúa como la etapa final de acondicionamiento de datos para la Estación de Control Terrestre. Su función es adaptar las variables estimadas y los estados lógicos del controlador a un formato legible para el piloto.

El bloque realiza las siguientes operaciones de procesamiento en tiempo real:

- **Conversión de Marco:** Transforma las coordenadas de navegación (NED) a estándares de aviación, invirtiendo la polaridad de la altitud y la velocidad vertical.
- **Cálculo de Velocidad:** Computa la velocidad terrestre (*Ground Speed*) mediante la norma vectorial de las componentes horizontales y aplica una zona muerta (*Deadband*) al variómetro para filtrar ruido en vuelo estacionario.
- **Gestión de Estado:** Multiplexa y formatea los datos críticos de operación (Voltaje de Batería, Modo de Vuelo y Estado de Armado) en un único bus de salida para su transmisión y visualización.

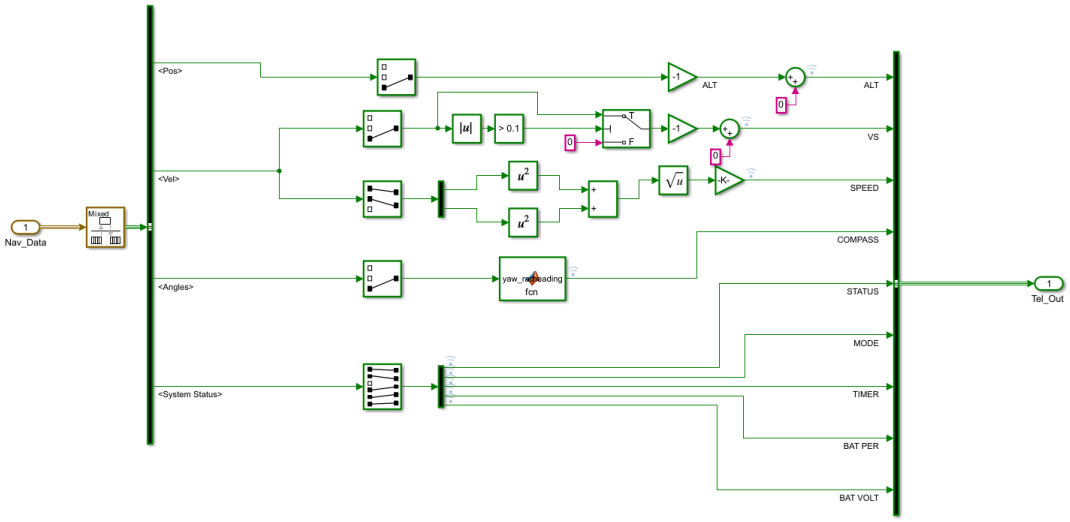

Fig. 3.2.5 *Lógica de acondicionamiento de señales para el sistema de Telemetría.*

3.3. Bloque Vehicle Interface

Este módulo representa la planta del sistema, encapsulando todos los fenómenos físicos y la dinámica del vehículo en el entorno simulado. A diferencia del controlador (que es discreto), este bloque se resuelve en tiempo continuo, ya que modela las ecuaciones diferenciales que rigen el movimiento en el mundo real.

Arquitectónicamente, mantiene la estructura de *Variant Subsystem*, lo que permite a futuro reemplazar este modelo matemático por una interfaz de hardware real (HIL) que conecte con los sensores y actuadores de un dron físico. Internamente, el modelo se descompone en siete subsistemas que interactúan entre sí:

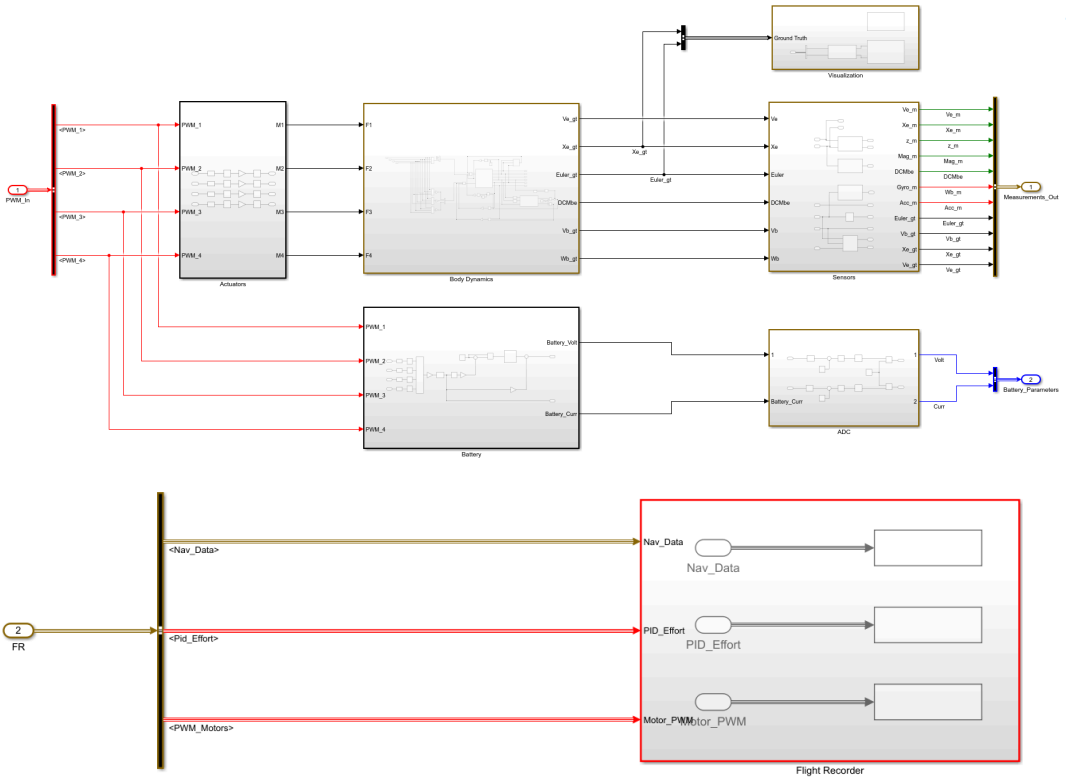

Fig. 3.3. *Arquitectura general de la planta física y flujo de señales continuas.*

3.3.1. Actuators

Modela la dinámica electromecánica del sistema de propulsión. Incluye la función de transferencia de los motores (retardo de primer orden), la relación cuadrática entre velocidad angular y empuje ($F \alpha \omega^2$), y las saturaciones físicas de los componentes.

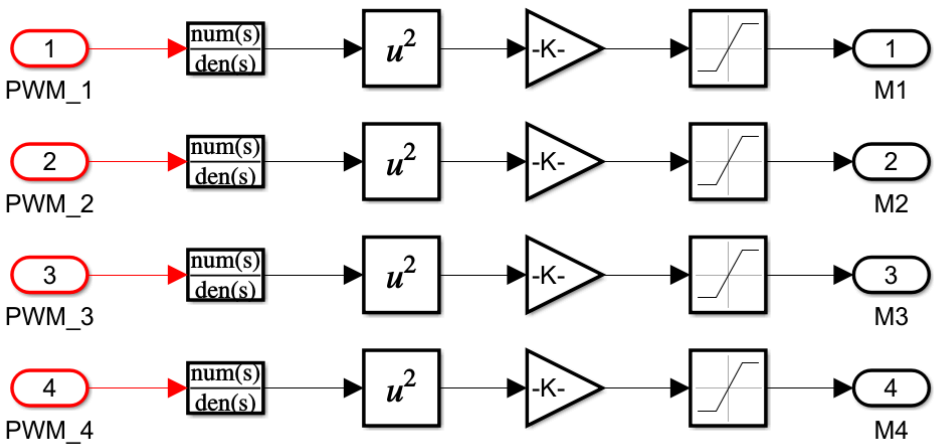

Fig. 3.3.1. *Subsistema Actuators: Dinámica de motores y saturación.*

3.3.2. Body Dynamics

Este subsistema constituye el núcleo físico de la simulación. Su funcionamiento inicia con el cálculo y la sumatoria vectorial de todas las fuerzas y momentos externos que actúan sobre el fuselaje:

- Las acciones de control provenientes de los motores.
- La fuerza de gravedad rotada al sistema de coordenadas del cuerpo (*Body Frame*).
- Las fuerzas de reacción del modelo de suelo (*Ground Model*) que impiden la caída libre bajo cota cero.
- Las cargas aerodinámicas, calculadas a partir de un modelo de viento que combina una componente estática y una dinámica (Turbulencia de Dryden), cuya intensidad y dirección son parametrizables en tiempo real desde un panel de control externo.

La resultante de estas fuerzas alimenta un bloque de 6 Grados de Libertad (6DOF) que integra las ecuaciones de movimiento de Newton-Euler para obtener, finalmente, todas las variables cinemáticas del vehículo (posición, velocidad lineal/angular y actitud).

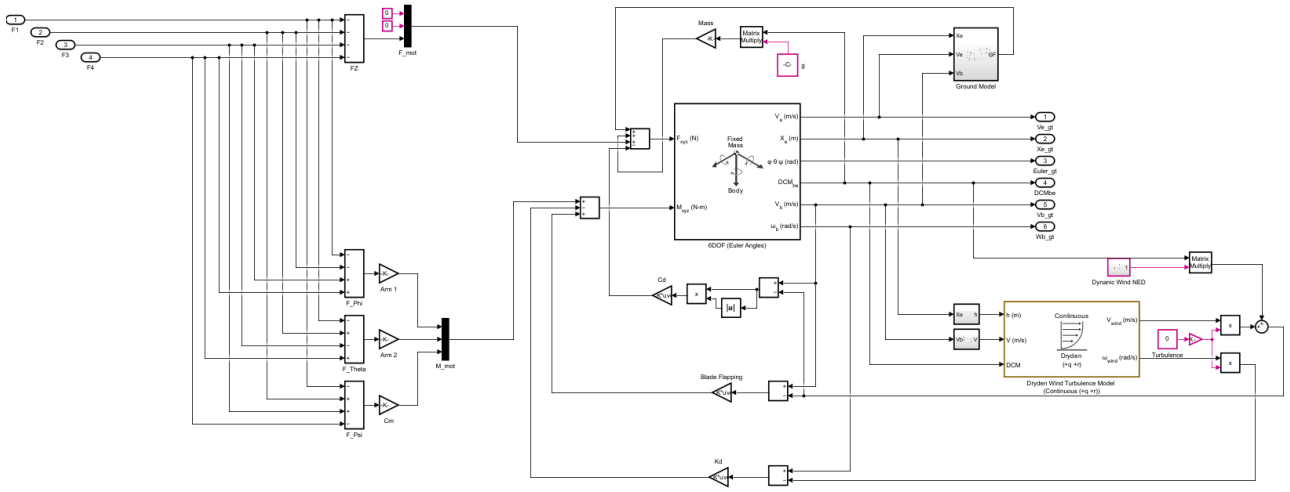

Fig. 3.3.2. *Subsistema Body Dynamics: Ecuaciones 6DOF, turbulencia y modelo de suelo.*

3.3.3. Sensors

Simula la cadena de adquisición de datos. Transforma los estados ideales ("Ground Truth") provenientes de la dinámica en señales realistas, inyectando imperfecciones típicas de sensores MEMS: Ruido Blanco Gaussiano Aditivo (AWGN), bias (sesgo) y deriva temporal (drift). Incluye modelos para GPS, Barómetro, Magnetómetro e IMU.

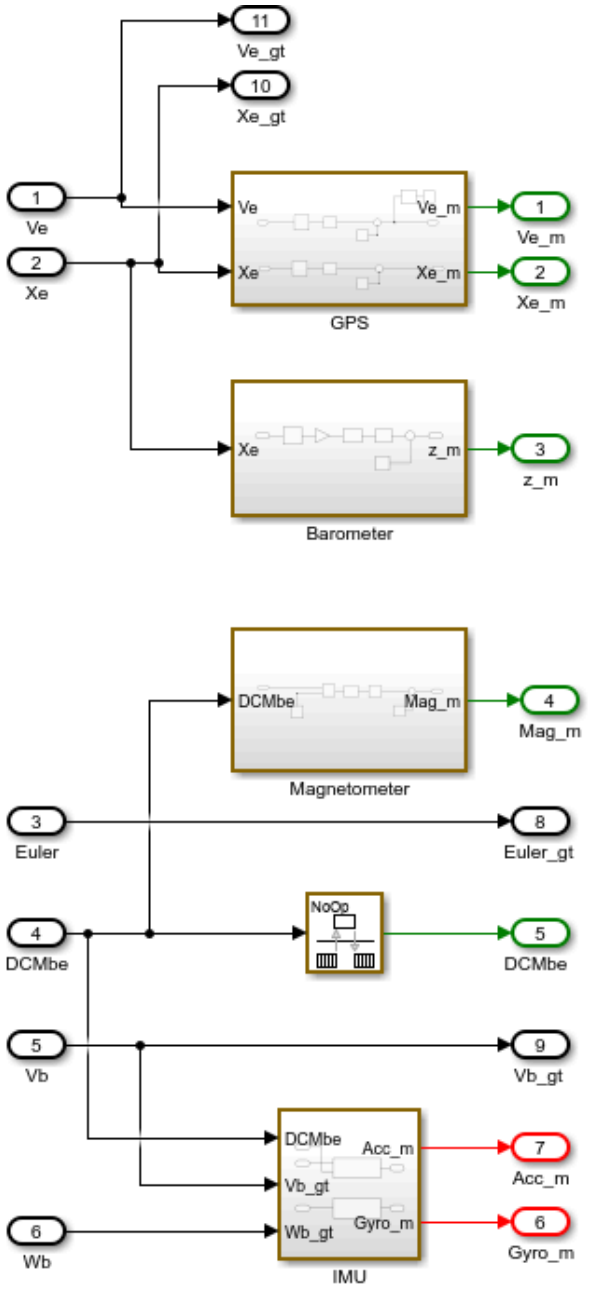

Fig. 3.3.3 *Subsistema Sensors: Modelado de ruido, bias y errores de instrumentación.*

3.3.4. Visualization

Este subsistema implementa una co-simulación gráfica de alta fidelidad utilizando el UAV Toolbox como puente de interfaz con Unreal Engine. Su función es recibir la telemetría de la simulación física y realizar la transformación de coordenadas necesaria (de la terna NED a las coordenadas de escena de Unreal) para sincronizar el movimiento del modelo dinámico con el entorno visual. Esto permite renderizar el vuelo en un escenario fotorrealista en tiempo real, facilitando la validación visual de las maniobras y la interacción con el terreno.

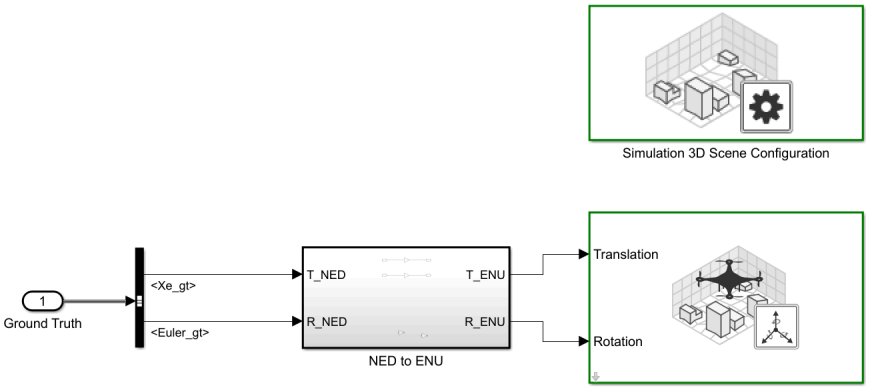

Fig. 3.3.4. *Subsistema Visualization: Transformación de coordenadas NED a ENU.*

3.3.5. Battery

Modelo de gestión energética del vehículo. Su lógica principal se centra en el cálculo preciso del voltaje en bornes. Para ello, el bloque integra en el tiempo el consumo de corriente instantáneo para determinar la capacidad drenada y el Estado de Carga (SoC). Posteriormente, utiliza este valor para interpolar el voltaje correspondiente mediante tablas de búsqueda (Lookup Tables) que representan la curva de descarga no lineal característica de la batería LiPo. Adicionalmente, incorpora la lógica del interruptor físico de corte general (Master Switch); al abrirse el circuito, fuerza la tensión de salida a cero, simulando la desconexión mecánica de la fuente.

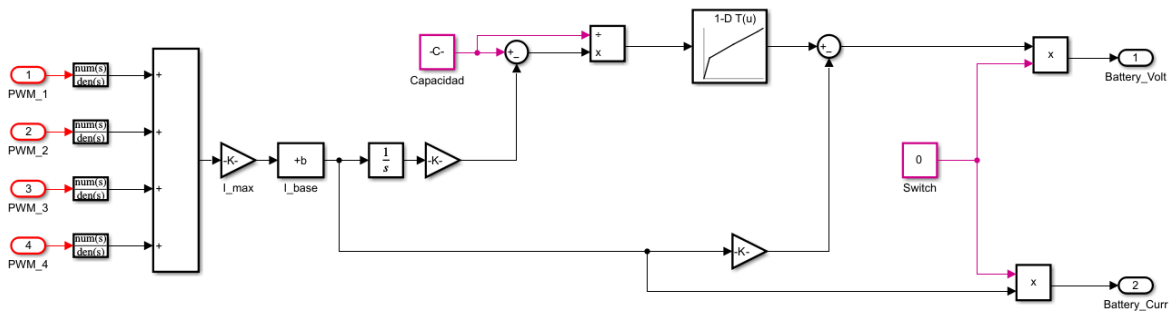

Fig. 3.3.5. *Subsistema Battery: Estimación de descarga basada en consumo de corriente.*

3.3.6. ADC (Analog-to-Digital Converter)

Simula la etapa de digitalización de las señales analógicas provenientes de los sensores de potencia. Replica las limitaciones físicas inherentes al hardware real introduciendo efectos de cuantización (resolución de bits) y discretización temporal (Zero-Order Hold), asegurando que las señales que llegan al controlador sean representaciones digitales fieles y no valores matemáticos ideales.

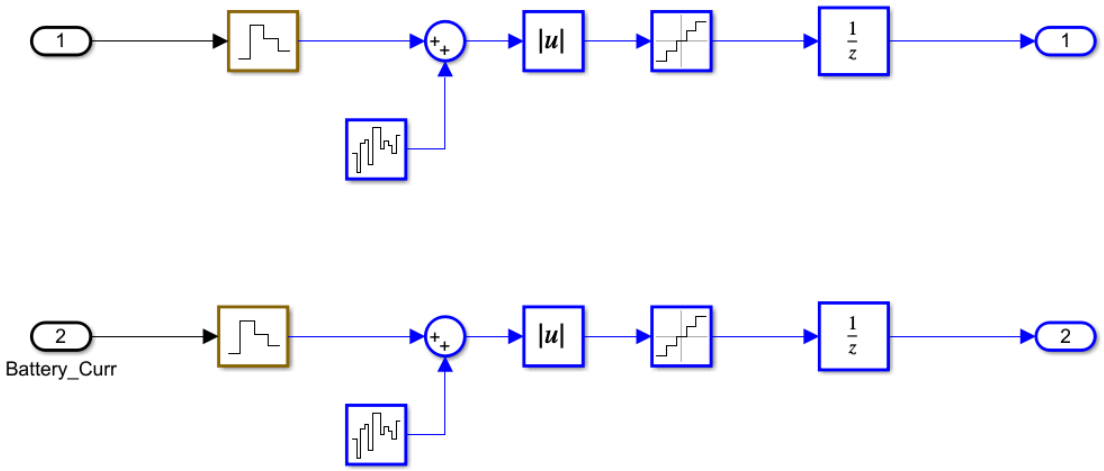

Fig. 3.3.6. *Subsistema ADC: Simulación de efectos de cuantización y discretización.*

3.3.7. Flight Recorder

Actúa como la "Caja Negra" del sistema. Almacena variables críticas de navegación, esfuerzos de control (PID) y señales PWM en el *Workspace* de MATLAB para su posterior análisis estático y validación de desempeño post-vuelo.

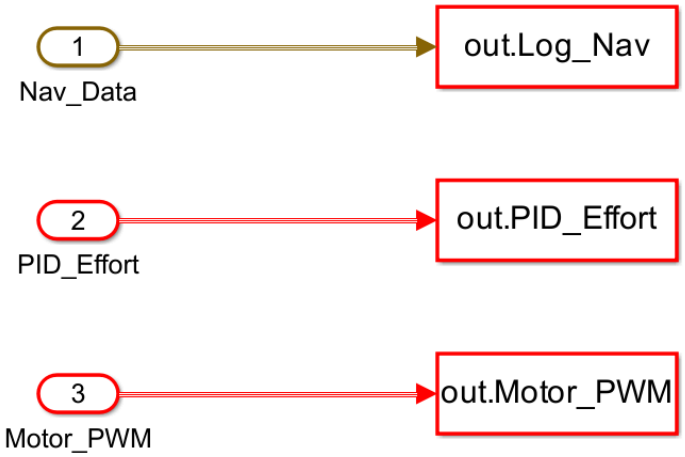

Fig. 3.3.7. *Subsistema Flight Recorder: Registro de datos para análisis post-vuelo.*

4. Paneles de Control

Para dotar al simulador de interactividad en tiempo real y facilitar la validación operativa sin depender de la instrumentación nativa de Simulink, se diseñó un conjunto de interfaces gráficas de usuario (GUI). Estas interfaces actúan como una capa de abstracción HMI (*Human-Machine Interface*), permitiendo al operador visualizar el estado del vehículo y manipular las condiciones del entorno de forma dinámica.

El sistema de visualización se divide en tres módulos funcionales:

4.1 Panel de Hardware

Diseñado para simular la interacción física con la electrónica del dron en tierra. Este panel permite gestionar el ciclo de energía y realizar las comprobaciones de pre-vuelo.

Controles e Indicadores:

- **Main Power Switch:** Interruptor que simula la conexión física de la batería LiPo. Al activarse, energiza los sistemas y dispara la lógica de inicialización.
- **Batt Volt Display:** Visualizador del voltaje de la batería. Permite verificar la tensión inicial y la caída de voltaje al armar los motores antes del despegue.
- **Status LED:** Indicador lumínico que replica el código de colores del LED a bordo, proporcionando confirmación visual del estado de armado o errores de calibración.

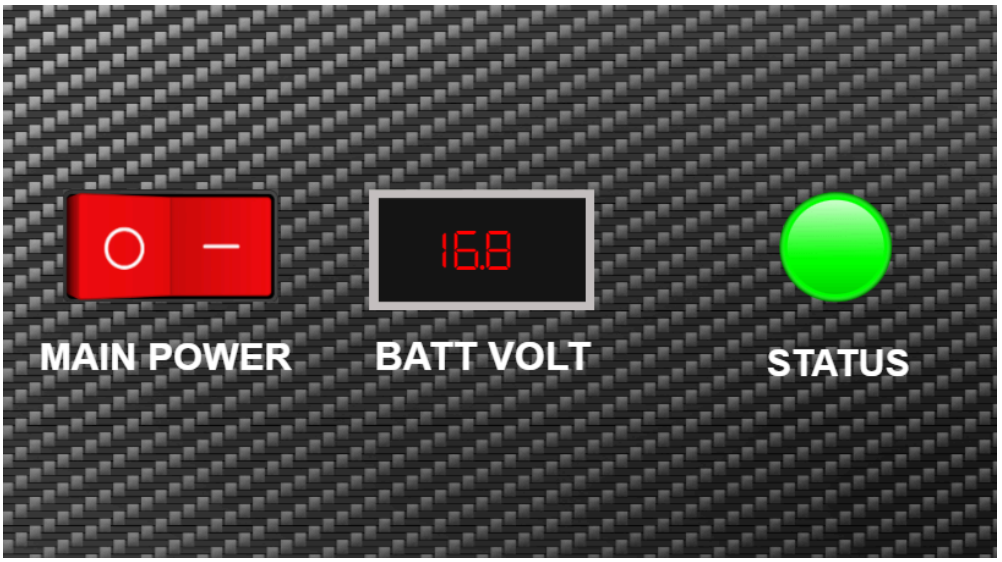

Fig 4.1. *Panel de Simulación de Hardware y Gestión de Energía.*

4.2. Panel de Telemetría

Este tablero de instrumentos digital centraliza la información crítica del vuelo, actuando como la interfaz primaria para el piloto. Su función es decodificar el bus de datos proveniente del controlador y presentar las variables de estado de forma clara e inmediata.

Elementos visualizados:

- **Instrumentación de Navegación:** Indicadores numéricos de Velocidad Terrestre (Speed), Altitud Barométrica (Alt) y Velocidad Vertical (VS).
- **Horizonte Artificial:** Indicador gráfico central que representa la actitud del vehículo (Roll y Pitch) respecto al horizonte.
- **Estado del Sistema:** Muestra el modo de vuelo activo (ACRO, STAB, GPS) y el estado de la máquina de estados (INIT, ARMED).
- **Gestión de Energía:** Monitoreo del porcentaje de batería restante y cronómetro de tiempo de misión.

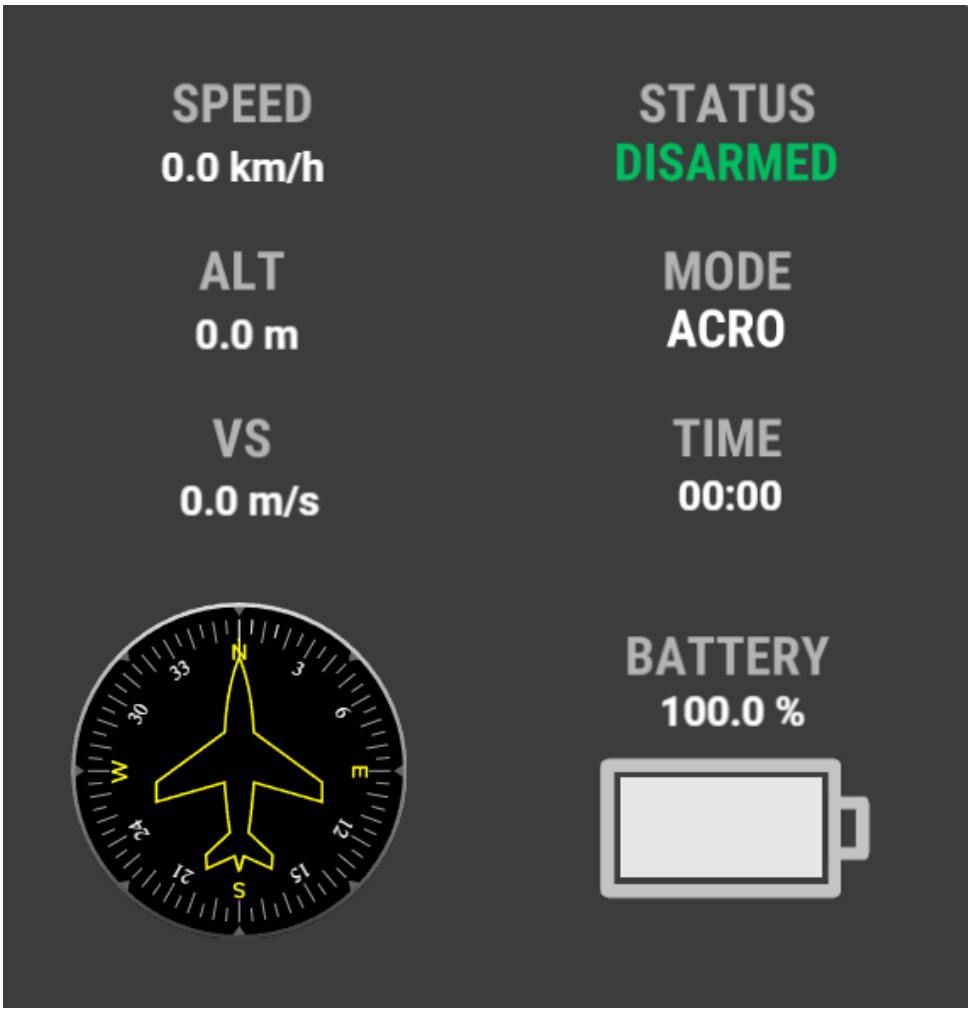

Fig 4.2. *Interfaz de Visualización de Telemetría e Instrumentación de Vuelo.*

4.3. Panel de Viento

Herramienta de ingeniería para la validación de robustez. Permite inyectar perturbaciones ambientales controladas al modelo físico, basándose en el modelo matemático de Turbulencia de Dryden.

Parámetros configurables:

- **Wind Speed (m/s):** Define la magnitud del vector de viento constante.
- **Wind Direction (°):** Establece la dirección de incidencia del viento respecto al marco de referencia inercial.
- **Turbulence (%):** Ajusta la ganancia del modelo Dryden, inyectando ráfagas estocásticas para evaluar la capacidad de rechazo de perturbaciones de los controladores PID.

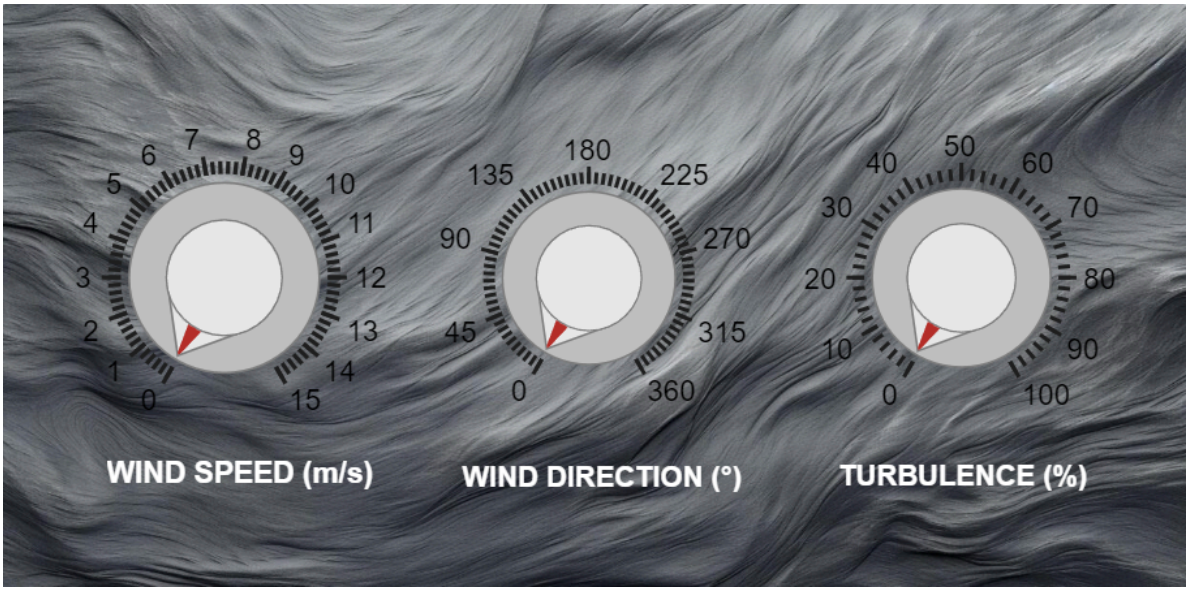

Fig 4.3. *Interfaz de Control de Perturbaciones Atmosféricas (Modelo Dryden).*

5. Script de Inicialización

Con el objetivo de evitar la inclusión de valores fijos (*hardcoding*) dispersos dentro de los bloques de Simulink y centralizar el control del sistema, se implementó una estrategia de carga de parámetros mediante un script de inicialización (*init_drone.m*). Este archivo se ejecuta automáticamente al abrir el proyecto, asegurando que todas las configuraciones estén siempre actualizadas.

Este script actúa como la referencia central del diseño, definiendo tres aspectos críticos:

1. **Arquitectura Temporal:** Cálculo automático de los tiempos de muestreo (T_s) para cada lazo del esquema Multi-rate (1000 Hz, 250 Hz, 100 Hz, 50 Hz).
2. **Modelo Físico del Vehículo:** Caracterización de los parámetros mecánicos y dinámicos (tensor de inercia, curvas de empuje de motores) y modelo de descarga de batería LiPo.
3. **Caracterización de Sensores y Entorno:** Configuración de las varianzas de ruido para la totalidad del paquete de sensores a bordo, así como las condiciones iniciales de la simulación.

*Nota: El código fuente completo de este script, con todos los valores y definiciones, se encuentra detallado en el **Anexo B**.*

6. Conclusiones y Futuros Proyectos

El desarrollo del presente trabajo se constituyó como un proceso de aprendizaje incremental. Es importante destacar que el punto de partida se caracterizó por un desconocimiento absoluto sobre el dominio general de los vehículos aéreos no tripulados y el entorno de desarrollo **Simulink**. Sin embargo, el **entusiasmo por la disciplina y la curiosidad técnica** actuaron como el motor fundamental para sostener la continuidad del proyecto. Esta motivación intrínseca permitió resolver cada dificultad técnica emergente e impulsó la incorporación paulatina de mayor fidelidad y nuevas funcionalidades al modelo.

Esta metodología constructiva derivó en una base sólida de competencias técnicas. Se logró desarrollar una valiosa intuición práctica sobre la **dinámica de los multicópteros** y los subsistemas de **Navegación y Control**, asimilando la lógica funcional y la relación "causa-efecto" de los algoritmos de estabilización. Este dominio práctico del sistema constituye ahora un sustento indispensable para abordar, con mayor profundidad y perspectiva, su posterior formalización matemática y teórica.

Finalmente, el aumento en la complejidad del diseño derivó en una necesidad logística no contemplada originalmente. La adopción de herramientas de control de versiones (**Git**) resultó ser un recurso auxiliar clave para mantener el orden, la trazabilidad y la integridad de las múltiples iteraciones del trabajo.

Dada la satisfacción personal y las competencias adquiridas durante este trabajo, se ha despertado un fuerte interés por extender la metodología de simulación hacia otros dominios de la ingeniería aeroespacial. Por ello, se proyectan las siguientes líneas de desarrollo prioritarias para continuar explorando durante este año:

- **Dinámica de Ala Fija:** Desarrollo de un simulador de vuelo completo que incluya sistemas de piloto automático, evaluando la posibilidad de utilizar como base la aeronave del anteproyecto de aerodinámica.

- **Sistemas Satelitales:** Simulación de mecánica orbital y control de actitud, implementando cuaterniones para una gestión robusta de la orientación y evitar singularidades (*gimbal lock*).
- **Continuidad del Multicóptero:** Avanzar hacia simulaciones *Hardware-in-the-Loop* (HIL) y *Processor-in-the-Loop* (PIL), con la intención de lograr eventualmente el ensamblaje de un dron físico funcional.
- **Vehículos Lanzadores:** Como objetivo a más largo plazo, abordar la simulación de la dinámica de ascenso y trayectoria de cohetes.

Anexo A: Gráficas

Fig A.1. *Ángulo de cabeceo estimado versus real.*

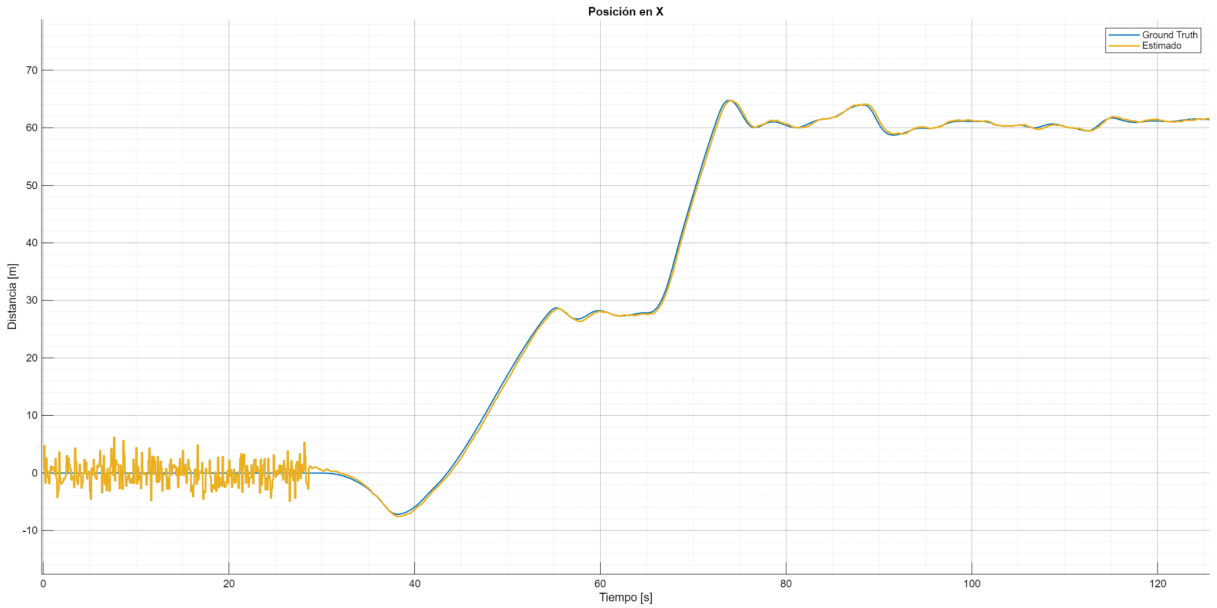

Fig A.2. *Posición en x estimada versus real.*

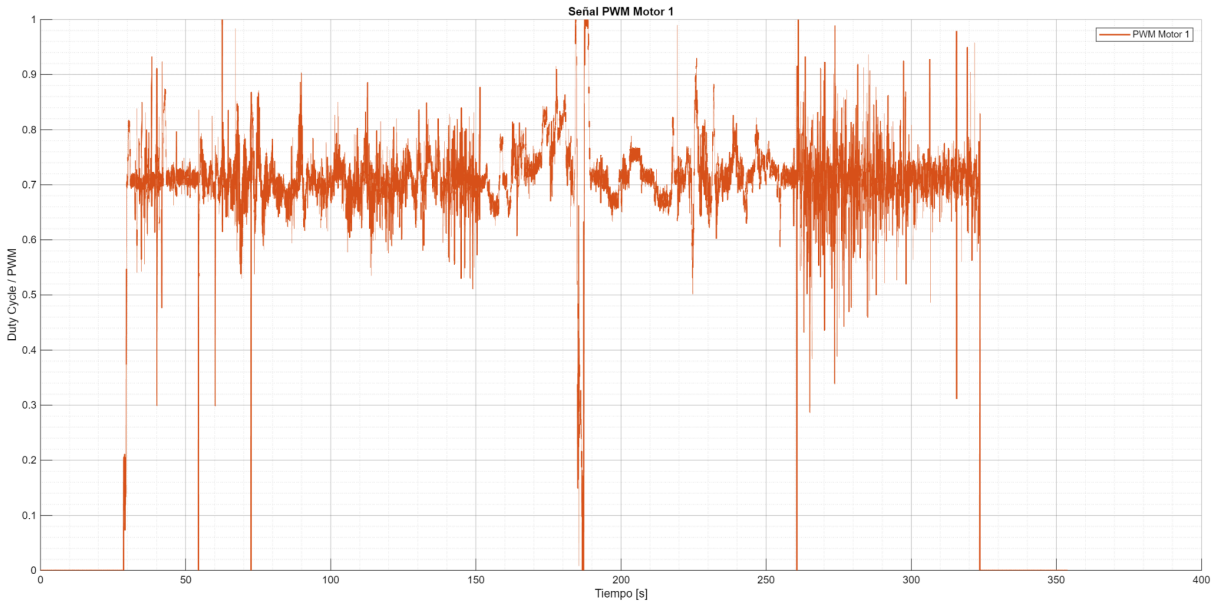

Fig A.3. *Señal de pwm del motor 1 para la evaluación de vibraciones.*

Anexo B: Algoritmos Implementados

Código B.1: *Script de inicialización y parametrización del sistema.*

```
%% ------------------------------------------------------------------------
%% SCRIPT DE INICIALIZACIÓN: PROYECTO DRONE
%% Descripción: Definición de parámetros físicos, arquitectura temporal
%%              y configuración de sensores para simulación.
%% ------------------------------------------------------------------------
clear; clc; close all;
disp('-> Cargando parámetros del sistema...');
%% 1. CONFIGURACIÓN DE ARQUITECTURA (MIL/SIL)
% ------------------------------------------------------------------------
% HIL_MODE: Selección de Planta (Modelo Matemático vs Hardware Real)
% 0 = Math Plant (Simulación)
% 1 = HIL (Sensores Reales)
HIL_MODE = 0;
% SIL_MODE: Implementación del Algoritmo de Control
% 0 = Model-in-the-Loop (Bloques Simulink)
% 1 = Software-in-the-Loop (Código C++ Generado)
SIL_MODE = 0;
fprintf('-> Configuración activa: HIL=%d | SIL=%d\n', HIL_MODE, SIL_MODE);
%% 2. DEFINICIÓN DE TIEMPOS DE MUESTREO (MULTI-RATE)
% ------------------------------------------------------------------------
% Frecuencias de operación [Hz]
Sim.Freq.Fisica       = 1000;  % Solver de ecuaciones dinámicas (ODE)
Sim.Freq.Control_Rate  = 1000;  % Lazo de Control de Tasa Angular
Sim.Freq.Control_Angle = 500;   % Lazo de Control de Actitud
Sim.Freq.Control_Pos   = 50;    % Lazo de Navegación/Posición
Sim.Freq.Supervisor    = 100;   % Lógica de Estados y Seguridad
Sim.Freq.Video         = 50;    % Tasa de actualización gráfica
% Frecuencias de muestreo de sensores [Hz]
Sim.Freq.Sensor_IMU    = 1000;
```

```
Sim.Freq.Sensor_Baro   = 50;
Sim.Freq.Sensor_Mag    = 50;
Sim.Freq.Sensor_GPS    = 10;
% Cálculo de Sample Times [s]
Sim.Ts_fisica   = 1 / Sim.Freq.Fisica;
Sim.Ts_control  = 1 / Sim.Freq.Control_Rate;
Sim.Ts_angle    = 1 / Sim.Freq.Control_Angle;
Sim.Ts_pos      = 1 / Sim.Freq.Control_Pos;
Sim.Ts_video    = 1 / Sim.Freq.Video;
Sim.Ts_logic    = 1 / Sim.Freq.Supervisor;
% Sample Times de Sensores
Sim.Ts_imu      = 1 / Sim.Freq.Sensor_IMU;
Sim.Ts_baro     = 1 / Sim.Freq.Sensor_Baro;
Sim.Ts_mag      = 1 / Sim.Freq.Sensor_Mag;
Sim.Ts_gps      = 1 / Sim.Freq.Sensor_GPS;
% Inicialización de Flags
Sim.Modo_Vuelo_Init = 0; % 0: Stabilize, 1: Acro
Sim.System_Ready    = 0;
%% 3. MODELO FÍSICO DEL VEHÍCULO
% -------------------------------------------------------------------
% 3.1. Inercia y Masa
Drone.Fisica.Masa = 2.0;         % [kg]
Drone.Fisica.g    = 9.81;        % [m/s^2]
Drone.Fisica.Ixx  = 0.035;       % [kg*m^2]
Drone.Fisica.Iyy  = 0.035;       % [kg*m^2]
Drone.Fisica.Izz  = 0.065;       % [kg*m^2]
Drone.Fisica.InertiaMatrix = diag([Drone.Fisica.Ixx, Drone.Fisica.Iyy, Drone.Fisica.Izz]);
% 3.2. Geometría (Configuración X)
Drone.Geom.d_x = 0.159;          % Brazo en eje X [m]
Drone.Geom.d_y = 0.159;          % Brazo en eje Y [m]
Drone.Geom.L   = sqrt(Drone.Geom.d_x^2 + Drone.Geom.d_y^2);
% 3.3. Aerodinámica
Drone.Aero.Cd = diag([0.030, 0.030, 0.060]);   % Coeficientes de Drag Traslacional
Drone.Aero.Kd = diag([0.01, 0.01, 0.05]);      % Coeficientes de Drag Rotacional
Drone.Aero.M_Flap = [0, -0.02, 0; 0.02, 0, 0; 0, 0, 0]; % Matriz de Flapping
% 3.4. Sistema de Propulsión
Drone.Motor.MaxThrust = 9.81;    % [N] Empuje máximo por motor
Drone.Motor.MinThrust = 0.2;     % [N] Empuje en idle
Drone.Motor.Ct = 1.0;            % Coeficiente de empuje (Adimensionalizado)
Drone.Motor.Cm = 0.015;          % Coeficiente de momento
%% 4. SISTEMA DE ENERGÍA Y SENSORES
% -------------------------------------------------------------------
% 4.1. Modelo de Batería (LiPo 4S)
Drone.Battery.Cells        = 4;
Drone.Battery.Capacity_Ah  = 5.0;
Drone.Battery.R_internal   = 0.012;      % [Ohms]
Drone.Battery.I_base       = 0.5;        % Corriente base (Aviónica)
Drone.Battery.I_max_motor  = 20.0;       % Corriente max por motor
Drone.Battery.MinVolts_Prot = 3.0 * Drone.Battery.Cells;
% Curvas de Descarga (Lookup Table)
Drone.Battery.DischargeCurve_Cap = [0, 0.1, 0.5, 0.9, 1.0];
Drone.Battery.DischargeCurve_Vol = [3.0, 3.5, 3.8, 4.1, 4.2] * Drone.Battery.Cells;
% 4.2. Caracterización de Ruido en Sensores (Varianza sigma^2)
% Sensor de Voltaje
Drone.Sensors.Voltage.Sigma     = 0.05;
Drone.Sensors.Voltage.NoisePower = (Drone.Sensors.Voltage.Sigma^2) * Sim.Ts_logic;
Drone.Sensors.Voltage.Bits      = 12;
Drone.Sensors.Voltage.V_ref     = 18.0;
Drone.Sensors.Voltage.Quantum   = Drone.Sensors.Voltage.V_ref / (2^Drone.Sensors.Voltage.Bits);
% Sensor de Corriente
Drone.Sensors.Current.Sigma     = 0.5;
Drone.Sensors.Current.NoisePower = (Drone.Sensors.Current.Sigma^2) * Sim.Ts_logic;
Drone.Sensors.Current.Bits      = 12;
```

```
Drone.Sensors.Current.I_max     = 100.0;
Drone.Sensors.Current.Quantum   = Drone.Sensors.Current.I_max / (2^Drone.Sensors.Current.Bits);
% IMU (Giroscopio y Acelerómetro)
Drone.Sensors.Gyro.Sigma      = 0.002;
Drone.Sensors.Gyro.NoisePower  = (Drone.Sensors.Gyro.Sigma^2) * Sim.Ts_imu;
Drone.Sensors.Accel.Sigma     = 0.1241;
Drone.Sensors.Accel.NoisePower = (Drone.Sensors.Accel.Sigma^2) * Sim.Ts_imu;
% Magnetómetro y Barómetro
Drone.Sensors.Mag.Sigma       = 0.002;
Drone.Sensors.Mag.NoisePower   = (Drone.Sensors.Mag.Sigma^2) * Sim.Ts_mag;
Drone.Sensors.Baro.Sigma      = 0.15;
Drone.Sensors.Baro.NoisePower  = (Drone.Sensors.Baro.Sigma^2) * Sim.Ts_baro;
% GPS
Drone.Sensors.GPS.Pos_Sigma     = 2.0;
Drone.Sensors.GPS.Vel_Sigma     = 0.1;
Drone.Sensors.GPS.Pos_NoisePower = (Drone.Sensors.GPS.Pos_Sigma^2) * Sim.Ts_gps;
Drone.Sensors.GPS.Vel_NoisePower = (Drone.Sensors.GPS.Vel_Sigma^2) * Sim.Ts_gps;
%% 5. VARIABLES DE ENTORNO Y CONDICIONES INICIALES
% -------------------------------------------------------------
Env.Viento.Vel_Base_NED   = [0; 0; 0];
Env.Viento.k_turb         = 0;
Env.Viento.Intensidad_Max = 15;
Env.Pos.Init_NED = [0, 0, -0.01];
%% 6. VISUALIZACIÓN
% -------------------------------------------------------------------------
try
  % Aplicación de paleta de colores para visualización de Sample Times
  miEstilo = simulink.sampletimecolors.Palette('EngineeringStyle');
  miEstilo.DiscreteSampleTimeColors = ["#FF0000", "#FFA500", "#0000FF", "#008000"];
  simulink.sampletimecolors.applyPalette(miEstilo);
catch
  % Fallback silencioso si la librería no está disponible
end
disp('-> Inicialización completada exitosamente.');
```

Código B.2: *Script de Procesamiento y Exportación de Telemetría.*

```
%% FLIGHT DATA LOGGING SYSTEM
% Description: Exports telemetry data to a specific folder, creating it if necessary.

% 1. Initialization and User Input
folder_name = 'telemetry_data'; % Nombre de la carpeta de destino
filename_base = input('Ingrese el nombre del archivo: ', 's');
if isempty(filename_base), filename_base = 'flight_telemetry'; end

% 2. Folder Management
% Check if the folder exists; if not, create it.
if ~exist(folder_name, 'dir')
   mkdir(folder_name);
   fprintf('Folder "%s" created successfully.\n', folder_name);
end

% 3. Telemetry Structure Definition
blocks = {'Log_Nav', 'PID_Effort', 'Motor_PWM', 'Ground_Truth', 'Measurements'};

% 4. Header Definitions
headers.Log_Nav = {'Vel_X','Vel_Y','Vel_Z','Pos_X','Pos_Y','Pos_Z',...
   'Roll_rad','Pitch_rad','Yaw_rad','p_rate','q_rate','r_rate',...
```

```
    'Status_ID','Flight_Mode','System_Ready','Batt_Percent','Batt_Volt',...
    'SP_Roll','SP_Pitch','SP_Yaw','SP_Mode','SP_Arm'};

headers.PID_Effort = {'Throttle_u', 'Roll_u', 'Pitch_u', 'Yaw_u'};
headers.Motor_PWM = {'Motor_1', 'Motor_2', 'Motor_3', 'Motor_4'};

headers.Ground_Truth = {'Vel_X_GT', 'Vel_Y_GT', 'Vel_Z_GT', ...
    'Pos_X_GT', 'Pos_Y_GT', 'Pos_Z_GT', ...
    'Roll_GT', 'Pitch_GT', 'Yaw_GT', ...
    'p_GT', 'q_GT', 'r_GT'};

headers.Measurements = {'Vel_X_m', 'Vel_Y_m', 'Vel_Z_m', ...
    'Pos_X_m', 'Pos_Y_m', 'Pos_Z_m', ...
    'Roll_m', 'Pitch_m', 'Yaw_m', ...
    'p_m', 'q_m', 'r_m'};

% 5. Execution Loop
fprintf('Starting Telemetry Export to /%s...\n', folder_name);

for i = 1:length(blocks)
    block_name = blocks{i};

    try
        raw_data = out.(block_name);
        t = raw_data.time;
        values = squeeze(raw_data.signals.values);

        if size(values, 2) == length(t), values = values'; end

        T = table(t, 'VariableNames', {'Time_s'});
        current_headers = headers.(block_name);
        [~, num_vars] = size(values);

        for j = 1:num_vars
            if j <= length(current_headers)
                T.(current_headers{j}) = values(:, j);
            else
                T.(sprintf('%s_ext_%d', block_name, j)) = values(:, j);
            end
        end

        % Construct Path: folder/filename_block.csv
        full_output_path = fullfile(folder_name, sprintf('%s_%s.csv', filename_base, block_name));

        writetable(T, full_output_path);
        fprintf('[SUCCESS] Exported: %s\n', full_output_path);

    catch ME
        fprintf('[WARNING] Failed to process block: %s. Error: %s\n', block_name, ME.message);
    end
end

fprintf('Export Process Finished.\n');
```

Código B.3: *Algoritmo de fusión de sensores para la estimación de actitud.*

```
function [bias_est, angle_est, DCM_be] = Attitude_Estimator(raw_accel_vec, gyro_rates, mag_body, system_ready)
% ATTITUDE_ESTIMATOR  Estimador de Orientación (AHRS) basado en EKF.
%
%   Implementa un Filtro de Kalman Extendido (EKF) de 6 estados para fusión sensorial.
%   Combina giroscopio (predicción), acelerómetro (corrección Roll/Pitch) y
%   magnetómetro (corrección Yaw con compensación de inclinación).
%
%   INPUTS:
%      raw_accel_vec : [ax; ay; az] en m/s^2 (Body Frame)
%      gyro_rates    : [p; q; r] en rad/s (Body Frame)
%      mag_body      : [mx; my; mz] gauss/uT (Body Frame)
%      system_ready  : Flag booleano (0=Calibración, 1=Vuelo)
%
%   OUTPUTS:
%      angle_est     : [Roll; Pitch; Yaw] en radianes (NED Frame)
%      DCM_be        : Matriz de Rotación 3x3 (Body -> Earth)
%      bias_est      : [bp; bq; br] Bias estimado del giroscopio

    % --- SYSTEM CONFIGURATION ---
    Ts = 0.001;  % Sample Time [s] (1 kHz)

    persistent x P Q R_acc R_mag ...
            accel_filtrado_prev is_frozen bias_captured ...
            calib_sum calib_count calib_bias_mem ...
            mag_offset_sum mag_yaw_offset

    % --- INITIALIZATION (First Run) ---
    if isempty(x)
        x = zeros(6, 1);
        P = diag([0.001, 0.0001, 0.001, 0.0001, 0.001, 0.0001]);
        Q = diag([0.0001, 1e-6, 0.0001, 1e-6, 0.0001, 1e-5]);
        R_acc = diag([500, 500]);
        R_mag = 5;

        accel_filtrado_prev = [0; 0; 0];
        is_frozen       = false;
        bias_captured   = false;
        calib_sum       = [0; 0; 0];
        calib_count     = 0;
        calib_bias_mem  = [0; 0; 0];
        mag_offset_sum  = 0;
        mag_yaw_offset  = 0;
    end

    % ========================================================================
    % 1. STATIC CALIBRATION & BIAS INITIALIZATION
    % ========================================================================
    if system_ready == 0
        if bias_captured
            calib_sum = [0;0;0]; calib_count=0; bias_captured=false; mag_offset_sum=0;
        end

        calib_sum = calib_sum + gyro_rates;

        mx = mag_body(1); my = mag_body(2);
        current_yaw_sample = -atan2(my, mx);
        mag_offset_sum = mag_offset_sum + current_yaw_sample;
```

```
    calib_count = calib_count + 1;
    calib_bias_mem = calib_sum / calib_count;
    mag_yaw_offset = mag_offset_sum / calib_count;

    angle_est   = [0; 0; 0];
    bias_est    = [0; 0; 0];
    DCM_be      = eye(3);
    return;
end

if ~bias_captured
    x(2) = calib_bias_mem(1);
    x(4) = calib_bias_mem(2);
    x(6) = calib_bias_mem(3);
    bias_captured = true;
end

% ========================================================================
% 2. PREDICTION STEP (Time Update)
% ========================================================================
p = gyro_rates(1);
q = gyro_rates(2);
r_corrected = gyro_rates(3) - x(6);

phi   = x(1);
theta = x(3);
tt = tan(theta);
ct = cos(theta);
if abs(ct) < 0.01, ct = 0.01; end

dot_phi   = p + (q * sin(phi) + r_corrected * cos(phi)) * tt;
dot_theta = q * cos(phi) - r_corrected * sin(phi);
dot_psi   = (q * sin(phi) + r_corrected * cos(phi)) / ct;

u = [dot_phi; dot_theta; dot_psi];

F = eye(6);
F(1,2) = -Ts; F(3,4) = -Ts; F(5,6) = -Ts;

G = zeros(6, 3);
G(1,1) = Ts; G(3,2) = Ts; G(5,3) = Ts;

x_pred = F * x + G * u;
P_pred = F * P * F' + Q;

% ========================================================================
% 3. CORRECTION STEP: ACCELEROMETER (Roll/Pitch)
% ========================================================================
acc_corrected = raw_accel_vec;
alpha = 0.1;
accel_filtrado = (acc_corrected * alpha) + (accel_filtrado_prev * (1 - alpha));
accel_filtrado_prev = accel_filtrado;

acc_mag = norm(accel_filtrado);

% Lógica de rechazo de aceleraciones dinámicas
if abs(acc_mag - 9.81) > 0.6
    is_frozen = true;
elseif abs(acc_mag - 9.81) < 0.5
    is_frozen = false;
end
```

```
    if is_frozen
        x_post_acc = x_pred;
        P_post_acc = P_pred;
    else
        ax = -accel_filtrado(1); ay = -accel_filtrado(2); az = -accel_filtrado(3);
        phi_acc   = atan2(ay, az);
        theta_acc = atan2(-ax, sqrt(ay^2 + az^2));

        H_acc = zeros(2, 6);
        H_acc(1,1) = 1; H_acc(2,3) = 1;

        y_acc = [phi_acc; theta_acc] - H_acc * x_pred;
        S_acc = H_acc * P_pred * H_acc' + R_acc;
        K_acc = P_pred * H_acc' / S_acc;

        x_post_acc = x_pred + K_acc * y_acc;
        P_post_acc = (eye(6) - K_acc * H_acc) * P_pred;
    end

    % ========================================================================
    % 4. CORRECTION STEP: MAGNETOMETER (Yaw)
    % ========================================================================
    phi_est   = x_post_acc(1);
    theta_est = x_post_acc(3);
    mx = mag_body(1); my = mag_body(2); mz = mag_body(3);

    Xh = mx * cos(theta_est) + my * sin(phi_est) * sin(theta_est) + mz * cos(phi_est) * sin(theta_est);
    Yh = my * cos(phi_est) - mz * sin(phi_est);

    psi_mag_raw = -atan2(Yh, Xh);
    psi_mag_corrected = psi_mag_raw - mag_yaw_offset;

    innov = psi_mag_corrected - x_post_acc(5);
    while innov > pi,  innov = innov - 2*pi; end
    while innov < -pi, innov = innov + 2*pi; end

    H_mag = zeros(1, 6);
    H_mag(1,5) = 1;

    S_mag = H_mag * P_post_acc * H_mag' + R_mag;
    K_mag = P_post_acc * H_mag' / S_mag;

    x_final = x_post_acc + K_mag * innov;
    P_final = (eye(6) - K_mag * H_mag) * P_post_acc;

    % ========================================================================
    % 5. OUTPUT GENERATION
    % ========================================================================
    x = x_final;
    P = P_final;

    psi_out = x(5);
    while psi_out > pi,  psi_out = psi_out - 2*pi; end
    while psi_out < -pi, psi_out = psi_out + 2*pi; end

    angle_est   = [x(1); x(3); psi_out];
    bias_est    = [x(2); x(4); x(6)];

    DCM_be = angle2dcm(psi_out, x(3), x(1), 'ZYX')';
end
```

Código B.4: *Algoritmo de fusión de sensores para la estimación de posición.*

```
function [pos_ned, vel_ned] = Position_Estimator(acc_body, DCM_be, gps_pos, gps_vel, baro_alt, system_ready)
% POSITION_ESTIMATOR  Estimador de Posición y Velocidad (INS/GPS)
%
%   Fusiona datos de Acelerómetro (IMU), GPS y Barómetro usando un EKF.
%   El sistema utiliza un marco de referencia NED (North-East-Down).
%   Recibe la orientación (DCM) calculada externamente para optimizar recursos.
%
%   INPUTS:
%       acc_body    : [ax; ay; az] Aceleración en m/s^2 (Body Frame)
%       DCM_be      : Matriz de Rotación 3x3 (Body -> Earth)
%       gps_pos     : [Lat; Lon; Alt] o [N; E; D] según configuración (m)
%       gps_vel     : [VelN; VelE] Velocidad GPS en m/s
%       baro_alt    : Altura barométrica [m] (Positivo hacia arriba)
%       system_ready : Flag de estado (0=Calibración, 1=Vuelo)
%
%   OUTPUTS:
%       pos_ned     : [Pn; Pe; Pd] Posición estimada [m]
%       vel_ned     : [Vn; Ve; Vd] Velocidad estimada [m/s]

    % --- SYSTEM CONFIGURATION ---
    Ts = 0.02;  % Tiempo de muestreo (50 Hz) - GPS suele ser más lento

    persistent x P Q R_base ...
            baro_offset calib_baro_sum calib_count ...
            initialized prev_gps_pos acc_lp

    % --- INITIALIZATION (First Run) ---
    if isempty(x)
        % Vector de Estado: [PosN, PosE, PosD, VelN, VelE, VelD]
        x = zeros(6,1);
        P = eye(6);

        % Q: Process Noise (Confianza en la física newtoniana)
        % q_pos: Baja incertidumbre en posición (no hay teletransportación)
        % q_vel: Mayor incertidumbre en velocidad (viento, ráfagas)
        q_pos = 0.0001;
        q_vel = 0.25;
        Q = diag([q_pos, q_pos, q_pos, q_vel, q_vel, q_vel].^2);

        % R: Measurement Noise (Desviación Estándar de sensores)
        sig_gps_pos = 3.0;  % Precisión GPS ~3 metros
        sig_baro_z  = 4.0;  % Barómetro ruidoso (filtrado agresivo)
        sig_gps_vel = 0.5;  % Precisión Velocidad GPS ~0.5 m/s

        % Matriz R Base (se seleccionará dinámicamente según disponibilidad)
        R_base = diag([sig_gps_pos, sig_gps_pos, sig_baro_z, sig_gps_vel, sig_gps_vel].^2);

        % Variables de calibración
        baro_offset     = 0;
        calib_baro_sum  = 0;
        calib_count     = 0;

        initialized     = false;
        prev_gps_pos    = [0;0;0];
        acc_lp          = [0;0;0];
    end
```

```
% ========================================================================
% 1. STATIC CALIBRATION & RESET (Ground Mode)
% ========================================================================
if system_ready == 0
   % Acumulamos lecturas del barómetro para encontrar el "Cero Local"
   calib_baro_sum = calib_baro_sum + baro_alt;
   calib_count = calib_count + 1;

   baro_offset = calib_baro_sum / calib_count;
   initialized = true;

   % Hard Reset del estado físico
   % Asumimos posición GPS actual como verdadera y velocidad cero
   x(1:2) = gps_pos(1:2); % Norte, Este
   x(3)   = 0;          % Abajo (Altura 0 relativa)
   x(4:6) = [0;0;0];     % Velocidades a 0

   prev_gps_pos = gps_pos;
   acc_lp       = acc_body;

   pos_ned = x(1:3);
   vel_ned = x(4:6);
   return;
end

% ========================================================================
% 2. PRE-PROCESSING (Body Frame)
% ========================================================================
% Filtro Paso Bajo (LPF) en acelerómetro para eliminar ruido de motor
alpha_acc = 0.1;
acc_lp = acc_lp * (1 - alpha_acc) + acc_body * alpha_acc;

% ========================================================================
% 3. ROTATION & GRAVITY COMPENSATION
% ========================================================================
% Rotar aceleración del cuerpo al mundo usando la matriz entrante (DCM)
acc_earth = DCM_be * acc_lp;

% Compensar gravedad (Restar 1G en el eje Z)
% En sistema NED: La gravedad es +9.81 m/s^2 (Hacia abajo).
% El acelerómetro mide -9.81 en reposo. Sumamos para cancelar.
acc_inertial_ned = acc_earth + [0; 0; 9.81];

% ========================================================================
% 4. PREDICTION STEP (Time Update)
% ========================================================================
% Modelo de arrastre lineal (Drag) para evitar velocidad infinita
k_drag = 0.05 * Ts;

% Matriz de Transición de Estado (F)
% Pos = Pos + Vel*Ts
% Vel = Vel * (1 - Drag) + Acc*Ts
F = eye(6);
F(1,4) = Ts; F(2,5) = Ts; F(3,6) = Ts;
F(4,4) = 1 - k_drag;
F(5,5) = 1 - k_drag;
F(6,6) = 1 - k_drag;

% Matriz de Control (G)
G = [0.5*Ts^2*eye(3);
       Ts*eye(3)];

x_pred = F * x + G * acc_inertial_ned;
```

```
    P_pred = F * P * F' + Q;

    % ========================================================================
    % 5. CORRECTION STEP (Measurement Update)
    % ========================================================================

    % Preparar medida Barométrica (Invertir signo para NED: Abajo es positivo)
    z_baro_ned = -1 * (baro_alt - baro_offset);

    % Detectar si el GPS trajo datos nuevos (es asíncrono)
    is_new_gps = any(gps_pos ~= prev_gps_pos);

    if is_new_gps
        % --- CASO A: Fusión Completa (GPS + Barómetro) ---
        prev_gps_pos = gps_pos;

        z_meas = [gps_pos(1); gps_pos(2); z_baro_ned; gps_vel(1); gps_vel(2)];

        H = zeros(5, 6);
        H(1,1) = 1; H(2,2) = 1; H(3,3) = 1; % Pos N, E, D
        H(4,4) = 1; H(5,5) = 1;            % Vel N, E

        R_curr = R_base;

    else
        % --- CASO B: Corrección Parcial (Solo Barómetro) ---
        z_meas = z_baro_ned;

        H = zeros(1, 6);
        H(1,3) = 1; % Solo observamos estado 3 (Pos D)

        R_curr = R_base(3,3);
    end

    % --- Actualización de Kalman (Forma de Joseph para estabilidad) ---
    y = z_meas - H * x_pred;       % Innovación (Medición - Predicción)
    S = H * P_pred * H' + R_curr;  % Covarianza de la Innovación
    K = (P_pred * H') / S;         % Ganancia de Kalman

    x = x_pred + K * y;            % Estado Corregido

    % Actualización de la Covarianza
    I = eye(6);
    P = (I - K * H) * P_pred * (I - K * H)' + K * R_curr * K';

    % ========================================================================
    % 6. OUTPUT GENERATION
    % ========================================================================
    pos_ned = x(1:3);
    vel_ned = x(4:6);
End
```