UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL ASISTIDO PARA PLATAFORMA AÉREA MULTI-ROTOR

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

FELIPE FERNÁNDEZ GUTIÉRREZ

PROFESOR GUÍA:
SR. JAVIER RUIZ-DEL-SOLAR

MIEMBROS DE LA COMISIÓN:
SR. MARCOS ORCHARD CONCHA
SR. HÉCTOR AGUSTO ALEGRÍA

SANTIAGO DE CHILE
2015

RESUMEN DE LA MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: FELIPE FERNÁNDEZ GUTIÉRREZ
FECHA: ENERO 2015
PROF. GUÍA: SR. JAVIER RUIZ DEL SOLAR

"DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE CONTROL ASISTIDO PARA PLATAFORMA AÉREA MULTI-ROTOR"

En esta Memoria de Título se diseñó y se implementó un sistema de control asistido para una plataforma de vuelo no tripulada (*Unmanned Aerial Vehicle* – UAV). Lo anterior se realizó con el propósito de simplificar el proceso de aprendizaje de operación radio-controlada a través de un mando de 4 canales. La plataforma utilizada correspondió a un quad-rotor, nave consistente en un cuerpo y cuatro hélices propulsoras. La tele-operación de UAVs constituye una tarea de lento aprendizaje debido a diversos factores como la dinámica de vuelo y la familiaridad del operario con los mandos. El principal objetivo del trabajo es el diseño y la implementación de un sistema que evalúe el desempeño del operario a través de un análisis de sus instrucciones y de la evolución de la nave, y que actúe a nivel de los controladores, facilitando en alguna medida el manejo de la nave. Este trabajo se realizó con el apoyo del Grupo de Automatización del Centro Avanzado de Tecnología para la Minería (AMTC).

La metodología de trabajo consistió, en primer lugar, en la integración de la plataforma aérea. Esta tarea incluyó el montaje de la estructura principal, la unión de todos los componentes mecánicos y electrónicos, el diagnóstico de los módulos utilizados, la instalación de un programa núcleo (*firmware*) de código abierto (APM: Copter), y la sintonización inicial de parámetros. Una vez lista la plataforma, se estudió el código fuente para determinar el tipo de asistente que se utilizaría. Posteriormente, se diseñó un sistema asistente de vuelo en base a pruebas de vuelo y un post-procesamiento de los datos realizado fuera de línea. Los sensores disponibles incluían las unidades de medición inercial, consistentes en acelerómetros y giróscopos, además de magnetómetros. Estos sensores no permitían una estimación adecuada de la posición de la nave, por lo que se consideró además la realización de simulaciones en base a los comandos entregados por el operario al momento dar la prueba. Se utilizó un simulador compatible con el código fuente utilizado y se escribió un programa en Python para enviar las órdenes al simulador mediante el protocolo de comunicación Mavlink y obtener coordenadas simuladas aproximadas. Posteriormente se creó un programa en MATLAB para analizar los resultados de ambas pruebas. Por otro lado, se modificó el código fuente, escrito en C++, creando un modo de vuelo que filtra las referencias entregadas a los controladores PID de alto nivel en función de dos parámetros que deben ser ajustados manualmente. Se realizaron pruebas con diferentes operarios para verificar tanto el funcionamiento del simulador como del asistente de vuelo.

Las simulaciones realizadas entregaron resultados imprecisos, con valores llegando sobre los 100 [m] para pruebas que no tuvieron desplazamientos superiores a los 3 [m]. Se consideró apropiado descartar su uso para la obtención de parámetros. Con ellos, se realizaron pruebas de operación para mostrar el efecto del asistente. Los operarios mostraron un grado de mejoría en las pruebas, específicamente pudiendo controlar la altitud de la plataforma con más estabilidad. Se obtuvieron oscilaciones lentas en los ángulos, sin embargo se mantuvieron dentro de rangos estables. Se concluyó que el asistente logra efectivamente intervenir de manera positiva en la estabilización de la nave, ayudando al control de los ángulos horizontales. Esto permite al usuario en primer lugar aprender a controlar la aceleración de la nave. Se proponen mejoras al trabajo, incluyendo la posible utilización de sensores de mayor precisión como GPS para la posición o láser o sonar para la altitud, lo que permitiría la posibilidad de asistir el control de aceleración y de obtener lecturas menos erráticas.

Agradecimientos

Este trabajo representa la culminación de muchos años de estudio, así como también una prueba de esfuerzo y perseverancia. A lo largo de este proceso he podido compartir con personas que me han apoyado de diferentes maneras, ya sea en las horas de estudio así como en los tiempos de descanso, tanto en Chile como en Francia. Quiero aprovechar esta instancia para agradecerles:

En primer lugar a mi familia, que me apoyaron en todo momento y vivieron conmigo todos los cambios que significó estudiar una carrera demandante. También les agradezco por el apoyo y la compañía que me brindaron estando fuera del país, la atención y la preocupación.

A Pamela, mi compañera durante gran parte de esta empresa, que aguantó tanto junto conmigo. Espero que este sea el inicio de muchas más cosas por venir, y que podamos apoyarnos mutuamente en todos nuestros proyectos y desafíos del futuro.

A mi profesor guía Javier Ruiz del Solar, por su orientación y dirección en este proceso, así como por su comprensión al momento de encontrar eventualidades, su buena disposición al momento de reunirnos a lo largo del trabajo, y por sobre todo por la oportunidad de trabajar en un área novedosa de la ingeniería en conjunto a investigadores de alto nivel.

A Rodrigo y en general a todo el equipo del Laboratorio de Robótica de Campo, por ayudarme en mis experimentos y en el desarrollo del trabajo.

A todos los FEOS, Santis, Nacho, Manolo, Vizuet, Choco, Juan, Diego, Marce. Ha sido un largo camino desde el colegio, sin falta de incidentes, pero ya estamos todos terminando, y el apoyo que nos hemos dado ha sido importantísimo.

A todos mis compañeros con los que compartí durante la carrera, en particular Jaime, Carlos, Diego, Pau, Pablo, con los que sacamos hartos ramos adelante en conjunto, todas las horas de clases y estudio fueron más entretenidas con ustedes.

A todo el equipo de Deportes Aventura, por los buenos momentos durante la carrera, espero nos reunamos de nuevo todos algún día cuando vuelvan al país para seguir la caída libre en equipo.

A Diego, Mati y Gabo, por el buen año que tuvimos con Rockover, espero que sigamos juntándonos y sacando adelante los proyectos que tenemos en mente.

A mis compañeros del Laboratorio de Control, en especial a David por ayudarme con algunos detalles del trabajo. También agradezco al profesor Marcos, quien siempre tuvo la mejor disposición para resolver dudas.

A todos los que estuvieron conmigo en Francia, en especial Rodrigo, Carlos, Pepe y Nico, grandes compañeros junto a los cuales viví excelentes experiencias, hicieron toda mi estadía mucho más llevadera.

A mis amigos de Francia, en especial Quentin, Alex y Richou, que me recibieron con la mejor disposición y me enseñaron tantas cosas.

A la familia Jacquemot, quienes me enseñaron cosas esenciales y me dieron una gran acogida en Francia.

Y en general a todos los que me brindaron apoyo en algún momento de mi carrera.

Tabla de Contenido

Índice de Tablas

Índice de Figuras

Capítulo 1: Introducción

1.1. Vehículos Aéreos No-Tripulados (UAV)

Los vehículos aéreos no tripulados (o *Unmanned Aerial Vehicles-UAV*) corresponden a aeronaves capaces de volar de manera autónoma, las cuales son controladas mediante algún sistema automático, o bien por un piloto en tierra. Estas naves pueden tener diversas funciones, como reconocimiento, vigilancia, transporte, entre otras. Existen diferentes clases de UAV, dependiendo de su arquitectura y sistema de propulsión.

Los *quad-rotor* pertenecen a la familia de multicópteros, los cuales funcionan mediante hélices que impulsan el vehículo y lo mantienen en el aire. Gracias al desarrollo tecnológico de los últimos años, actualmente existen diversas configuraciones estándar para el diseño y construcción de estos vehículos. La investigación sobre la implementación de esquemas de control más avanzados en este tipo de aeronaves es reciente. Por otro lado, generalmente la experiencia de aprendizaje de manejo de dichas naves presenta dificultades importantes para los usuarios inexpertos, siendo los factores más relevantes la dinámica de estas naves y su respuesta a instrucciones del operador.

1.2. Estado del Arte

La investigación en el área de UAV se ha desarrollado en gran medida desde los años 90, gracias a la miniaturización de la tecnología. Se han realizado muchos modelos y estudios de distintos tipos de naves, incluido el quad-rotor. Las estrategias de control se han aplicado desde inicios de los años 2000, junto con la posibilidad de construcción e implementación de *hardware* y *software* más potente, como microcontroladores capaces de manejar millones de instrucciones por segundo, así como la reducción en el tamaño de las baterías, principalmente de ion-litio. Desde el año 2005 muchos investigadores se han centrado en investigar el quad-rotor, debido a su robustez, maniobrabilidad y bajo costo.

Las publicaciones recientes se centran principalmente en:

- La aplicación de nuevas estrategias de control, principalmente no lineal.
- La creación y validación de modelos en distintos tipos de software como MATLAB Simulink u otros de tipo CAD.
- La implementación de diferentes estrategias de creación y seguimiento de trayectorias (*path-finding*), incluyendo detección y evasión de obstáculos.
- El control basado en sensores de visión (cámaras).

1.3. Investigaciones actuales del AMTC ligados al uso de UAV

El AMTC (Centro Avanzado de Tecnología para la Minería, por sus siglas en inglés) está dividido en variados grupos de trabajo que integran numerosos investigadores y tesistas. Las diferentes líneas de investigación son:

- Exploración y Modelamiento de Yacimientos
- Diseño y Planificación Minera
- Procesamiento de Minerales y Metalurgia Extractiva
- Automatización en Minería
- Agua y Sustentabilidad Mineral

Dentro de estos equipos de trabajo, el grupo de automatización abarca numerosos ámbitos de esta disciplina, como robótica, reconocimiento de patrones, procesamiento de imágenes, entre otros, los cuales integra en el sector minero. En esta sección se resumen algunos proyectos que podrían utilizar UAVs en su desarrollo o implementación.

1.3.1. Modelamiento y Mapeo 3D de Minas

El objetivo principal de este proyecto es poder estimar perfiles de superficie de rocas y la construcción de mapas en 3D de manera remota. Esto se lleva a cabo mediante el uso de vehículos todo-terreno con sensores como láser, cámara y radar en conjunto con un posterior procesamiento de datos que permita obtener datos sin ruido.

Hasta ahora, el proyecto se ha centrado en la implementación en un vehículo Husky A200 de diversos sensores y el procesamiento necesario para estimar la localización de objetos y su número, lo que permite a su vez extraer información de perfiles de superficie de terreno y mapeo de minas. La implementación de las técnicas diseñadas en UAVs ampliaría sus aplicaciones a sectores inaccesibles para vehículos terrestres, como túneles con obstáculos ineludibles o minas de acceso vertical, entre otros.

1.3.2. Sistema de monitoreo para seguridad de trabajadores en la minería con aprendizaje automático

Con este proyecto se busca desarrollar un sistema de monitoreo basado en cámaras de video y software de procesamiento de imágenes que permita reducir los riesgos laborales de los operadores en plantas de procesos del sector minero. Esto se realiza mediante alarmas preventivas en conjunto con modelos de aprendizaje continuo.

Se ha propuesto un sistema que monitorea áreas específicas de las faenas, pudiendo reconocer objetos, personas, y determinar condiciones de operación riesgosas para los trabajadores. Para ello, se crea una extensa base de datos de imágenes que pueda entrenar al sistema. Por otro lado, se desarrolla una interfaz de alarmas y alertas, además de integrar información de diversas cámaras. Hasta el momento, los resultados indican que el sistema es capaz de detectar personas y vehículos con precisión.

La posible integración de cámaras montadas en UAVs podría entregar información adicional al sistema, con nuevas imágenes diferentes a las entregadas por cámaras fijas. Esto puede ser de utilidad en particular en situaciones o lugares donde las imágenes disponibles no son suficientes, o para dar redundancia al sistema en casos de utilización de maquinaria particularmente peligrosa.

1.4. Objetivos y Estructura de Trabajo

1.4.1. Objetivos Generales

El objetivo general de este trabajo es el diseño y la implementación de un sistema de control asistido que se adapte al usuario. Este sistema debe facilitar el aprendizaje de la manipulación de la nave, a la vez procurando mantener la integridad de la instrumentación y evitando en lo posible accidentes debido a inexperiencia del operador.

1.4.2. Objetivos Específicos

En este trabajo, los objetivos específicos considerados son:

- Estudiar métodos de control y estrategias de sensorización aplicables al quad-rotor.
- Definir el *hardware*, incluyendo sensores y controlador, que se utilizará para lograr un control robusto.
- Integrar un quad-rotor, incluyendo un sistema electrónico y de control *open-source*.
- Diseñar un sistema de control asistido que facilite el aprendizaje y la manipulación del quad-rotor.
- Implementar dicho sistema en el *software* de control del quad-rotor.
- Realizar pruebas que muestren el resultado de la implementación y comparar con la versión original sin sistema de control asistido.

1.4.3. METODOLOGÍA

Para el desarrollo de este trabajo, se consideraron dos grandes etapas, consistiendo la primera en la integración de una plataforma aérea *quad-rotor*, para luego proceder al diseño e implementación del sistema de control asistido. Las etapas seguidas se listan a continuación:

Etapa Preliminar: Recopilación Bibliográfica

- Estudio de UAVs y de quad-rotors, definiendo sus componentes esenciales.
- Estudio de controladores usados en quad-rotors.
- Estudio de *software* y *hardware* asociado.
- Estudio del estado del arte en aplicaciones de UAV.

Primera Etapa: Integración de la plataforma

- Definición de componentes a utilizar y obtención de elementos faltantes por parte del Laboratorio, incluyendo un estudio económico.
- Integración de la mecánica y electrónica involucrada en la plataforma considerada, incluyendo la instalación de *software* y *firmware*.
- Pruebas preliminares, calibración, sintonización y reajustes necesarios para lograr un vuelo estable del quad-rotor.

Segunda Etapa: Diseño e Implementación del Sistema de control asistido

- Estudio del código ya implementado en la plataforma.
- Diseño del sistema de control asistido.
- Implementación en *firmware* y *software*.
- Pruebas finales en la plataforma.

1.4.4. ESTRUCTURA DE LA MEMORIA

Este informe se estructura de la siguiente manera. En primer lugar, el Capítulo 2 sirve al lector para contextualizar tanto los conceptos teóricos mencionados, así como la problemática que se aborda. En éste además se presenta la plataforma a utilizar, el quad-rotor, con su modelo y sus componentes. A continuación, el Capítulo 3 muestra la implementación del trabajo realizado. En esta sección se detalla en primer lugar el proceso de integración de la plataforma, detallando los problemas encontrados y las soluciones escogidas. A continuación se describe el proceso de sintonización de los controladores, mientras que las siguientes secciones abordan el diseño y la implementación del sistema de control asistido, presentando el problema a abordar y la solución escogida. El Capítulo 4 presenta los resultados tanto de las pruebas preliminares del quad-rotor, así como de las pruebas comparativas una vez implementado el asistente. Finalmente en el Capítulo 5 se presentan las conclusiones generales del trabajo realizado y se discuten posibles mejoras y trabajo futuro al respecto.

Capítulo 2: Marco Teórico

El presente capítulo sirve al lector para familiarizarse con los conceptos teóricos que se mencionan en este trabajo, así como mostrar un estudio tanto de la plataforma a utilizar como del problema que se desea abordar, poniendo énfasis en la problemática existente. En primer lugar, en la Sección 2.1 se aborda el modelo matemático que rige la dinámica de los quad-rotor con un enfoque basado en control de sistemas. A continuación en la Sección 2.2 se detallan los componentes básicos que actualmente se presentan en las configuraciones estándar, al igual que los diferentes tipos de sensores que deben estar incorporados. Luego la Sección 2.3 presenta algunos firmwares de código abierto (*open-source*) que se encuentran disponibles en la red con cierto respaldo por parte de desarrolladores y comunidad, mostrando sus características más importantes. Finalmente, la Sección 2.4 detalla el tipo de control que se utiliza en el *firmware* ArduPilotMega, mientras que en la Sección 2.5 se aborda el problema de autonomía y toma de decisiones en vehículos aéreos autónomos.

2.1. Modelo Matemático del Quad-Rotor

El **quad-rotor** corresponde a un multi-cóptero de 4 rotores y 4 hélices, los cuales están dispuestos en los extremos de brazos de igual largo que se encuentran alrededor del cuerpo principal. Las hélices frontal y trasera, (H_f, H_b), giran en un sentido, mientras que las hélices derecha e izquierda, (H_r, H_l), giran en el sentido opuesto. Esto resulta en una cancelación de efectos giróscopos y torques aerodinámicos en estado de vuelo compensado [**1**].

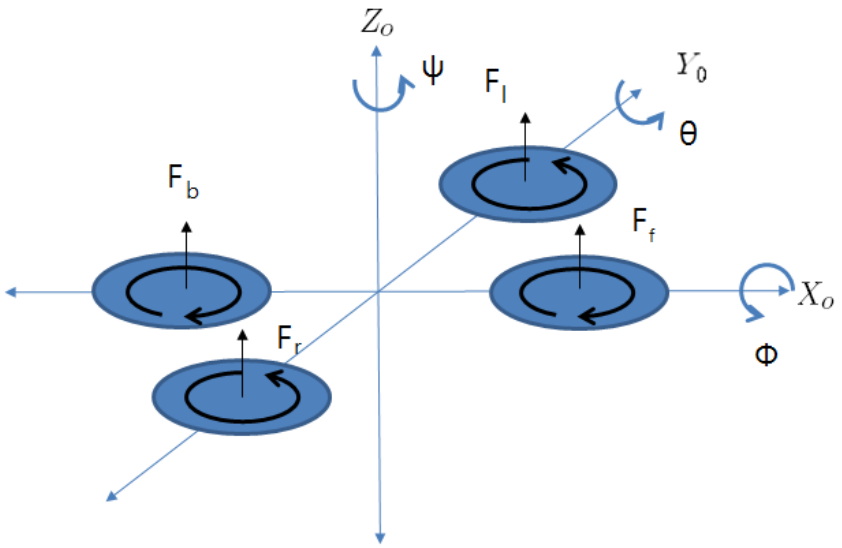


Figura 2.1: Diagrama esquemático de un quad-rotor, indicando los ejes de rotación y las fuerzas ejercidas por los rotores.

Las traslaciones a lo largo de los ejes $(\hat{x}_q, \hat{y}_q, \hat{z}_q)$ así como las rotaciones (ϕ, θ, ψ) a lo largo de ellos se obtienen al variar las velocidades angulares ω_i de los 4 motores. Para entender la dinámica del vehículo, se debe realizar un análisis físico contemplando la energía cinética y potencial del cuerpo, así como las fuerzas internas y externas que actúan sobre él. El análisis contemplado en [**1**] toma un enfoque basado en las ecuaciones de Euler-Lagrange de cálculo de variaciones, y será descrito de forma resumida a continuación.

Las coordenadas generalizadas del quad-rotor se denotan

$$q = (x, y, z, \phi, \theta, \psi) \in \mathbb{R}^6 \qquad (2.1)$$

donde (x, y, z) es la posición del centro de masa de la nave con respecto a una referencia fija F_0, y los ángulos (ϕ, θ, ψ) se denominan *ángulos de Euler*, llamados ángulo de **cabeceo** o **elevación** (*pitch*), de **alabeo** (*roll*) y de **dirección** o **guiñada** (*yaw*), respectivamente. Estos ángulos representan la orientación de la nave respecto a la referencia F_0. La energía cinética del cuerpo se puede escribir en función de las coordenadas traslacionales y rotacionales como sigue

$$E_c = E_{c_{tr}} + E_{c_{rot}} = \frac{m}{2}\dot{\xi}^T\dot{\xi} + \frac{1}{2}\dot{\eta}^T\mathbb{I}\dot{\eta} \qquad (2.2),$$

donde $\xi = (x, y, z)$ son las coordenadas traslacionales y $\eta = (\phi, \theta, \psi)$ son las coordenadas rotacionales; m es la masa del vehículo; $\mathbb{I}$ representa la matriz de inercia expresada en las coordenadas generalizadas. Por otro lado, la energía potencial corresponde simplemente al potencial gravitacional

$$U = mgz \qquad (2.3)$$

Con esto, la expresión del Lagrangiano queda dada por

$$\begin{aligned} L(q, \dot{q}) = &\quad E_{c_{tr}} + E_{c_{rot}} - U \\ = &\quad \frac{m}{2}\dot{\xi}^T\dot{\xi} + \frac{1}{2}\dot{\eta}^T\mathbb{I}\dot{\eta} - mgz \end{aligned} \qquad (2.4)$$

El Lagrangiano representa la dinámica del sistema en términos de su energía. Para llegar a las ecuaciones de movimiento, se debe resolver la ecuación de Euler-Lagrange, la cual está dada por

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q} = \Im \qquad (2.5)$$

donde $\Im = (F_\xi, \tau)$ es el vector de fuerzas generalizadas; F_ξ corresponde a la fuerza traslacional aplicada a la nave y τ es el vector de torques generalizados. Reemplazando (2.4) en (2.5) y considerando el desacoplamiento de las coordenadas traslacionales ξ y rotacionales η se obtiene el sistema de ecuaciones

$$m\ddot{\xi} + \begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix} = \quad F_\xi \qquad (2.6),$$

$$\mathbb{I}\ddot{\eta} + \dot{\mathbb{I}}\dot{\eta} - \frac{1}{2}\frac{\partial}{\partial \eta}(\dot{\eta}^T\mathbb{I}\dot{\eta}) = \quad \tau \qquad (2.7)$$

La fuerza F_ξ aplicada al quad-rotor corresponde a la suma de las fuerzas de cada rotor, f_i, multiplicada por la matriz de transformación que representa la rotación de la nave con respecto a F_0

$$\begin{aligned} F_\xi = &\quad R\hat{F} \\ = &\quad \begin{pmatrix} c_\theta c_\psi & s_\psi s_\theta & -s_\theta \\ c_\psi s_\theta s_\phi - s_\psi c_\phi & s_\psi s_\theta s_\phi + c_\psi c_\phi & c_\theta s_\phi \\ c_\psi s_\theta c_\phi + s_\psi s_\phi & s_\psi s_\theta c_\phi - c_\psi s_\phi & c_\theta c_\phi \end{pmatrix}\begin{pmatrix} 0 \\ 0 \\ u \end{pmatrix} \end{aligned} \qquad (2.8),$$

donde c_χ y s_χ denotan $\cos(\chi)$ y $\sin(\chi)$ respectivamente, y u representa la acción de control en el eje $\hat{z}_q$ relativo a la nave

$$u = f_1 + f_2 + f_3 + f_4 \qquad (2.9),$$

$$f_i = k_i\omega_i^2\ , i = 1, \dots, 4 \qquad (2.10)$$

Las constantes k_i representan la proporcionalidad entre el cuadrado de la velocidad angular del *i*-ésimo rotor ω_i y la fuerza f_i que éste ejerce. Por otro lado, los torques generalizados pueden ser escritos como

$$\tau = \begin{pmatrix} \tau_\psi \\ \tau_\theta \\ \tau_\phi \end{pmatrix}, \qquad (2.11),$$

donde los torques en torno a cada eje son

$$\tau_\psi = \sum_{i=1}^{4} \tau_{M_i}$$
$$\tau_\theta = (f_2 - f_4)l$$
$$\tau_\phi = (f_3 - f_1)l,$$

con τ_{M_i} el torque producido por el *i*-ésimo rotor, y *l* el largo de los brazos.

Dadas las ecuaciones (2.8) y (2.11), el sistema de ecuaciones (2.6) y (2.7) puede ser reescrito como sigue

$$m\ddot{\xi} = u \begin{pmatrix} -s_\theta \\ c_\theta s_\phi \\ c_\theta c_\phi \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ -mg \end{pmatrix}, \qquad (2.12),$$
$$\mathbb{I}\ddot{\eta} = -C(\eta, \dot{\eta})\dot{\eta} + \tau, \qquad (2.13),$$

donde el término $C(\eta, \dot{\eta})$ se denomina término de Coriolis y agrupa los efectos giroscópicos y centrífugos asociados con la dependencia de $\mathbb{I}$ con respecto a las coordenadas rotacionales η. Finalmente, al considerar una simplificación con el cambio en las variables de entrada

$$\tau = C(\eta, \dot{\eta})\dot{\eta} + \mathbb{I}\tilde{\tau} \Leftrightarrow \tilde{\tau} = \mathbb{I}^{-1}(\tau - C(\eta, \dot{\eta})\dot{\eta}) \qquad (2.14),$$

se obtienen las ecuaciones de movimiento del quad-rotor en función de la fuerza u aplicada a la nave, y de los torques generalizados $\tilde{\tau}$ en torno a los ejes del quad-rotor ($\hat{x}_q, \hat{y}_q, \hat{z}_q$):

$$m\ddot{x} = -u \sin\theta \qquad (2.15),$$
$$m\ddot{y} = u \cos\theta \sin\phi \qquad (2.16),$$
$$m\ddot{z} = u \cos\theta \cos\phi - mg \qquad (2.17),$$
$$\ddot{\psi} = \tilde{\tau}_\psi \qquad (2.18),$$
$$\ddot{\theta} = \tilde{\tau}_\theta \qquad (2.19),$$
$$\ddot{\phi} = \tilde{\tau}_\phi, \qquad (2.20)$$

Este sistema representa la dinámica del quad-rotor, en donde las entradas del sistema corresponden a las acciones de control $(u, \tilde{\tau}_\psi, \tilde{\tau}_\theta, \tilde{\tau}_\phi)$, y las salidas corresponden a las coordenadas traslacionales (x, y, z) y de orientación (ϕ, θ, ψ). Claramente, existe una diferencia entre el número de entradas (o variables **manipuladas**) y el número de salidas del sistema (también llamadas variables **controladas**). Es posible, mediante una manipulación de las ecuaciones descrita en [**1**], establecer una relación directa entre las variables de entrada y salida que se indican en la Tabla 2.1.

Las estrategias de control se diseñan en base a estas dependencias. Para ello, se deben considerar los esquemas de **lazo cerrado** que contemplan los elementos que se muestran en la Figura 2.2.

Tabla 2.1: Relación entre variables manipuladas y controladas que es posible aprovechar para el diseño de controladores.

Variable Manipulada	Variable Controlada Dependiente
u	z
$\tilde{\tau}_{\psi}$	ψ
$\tilde{\tau}_{\theta}$	x
	θ
$\tilde{\tau}_{\phi}$	y
	ϕ

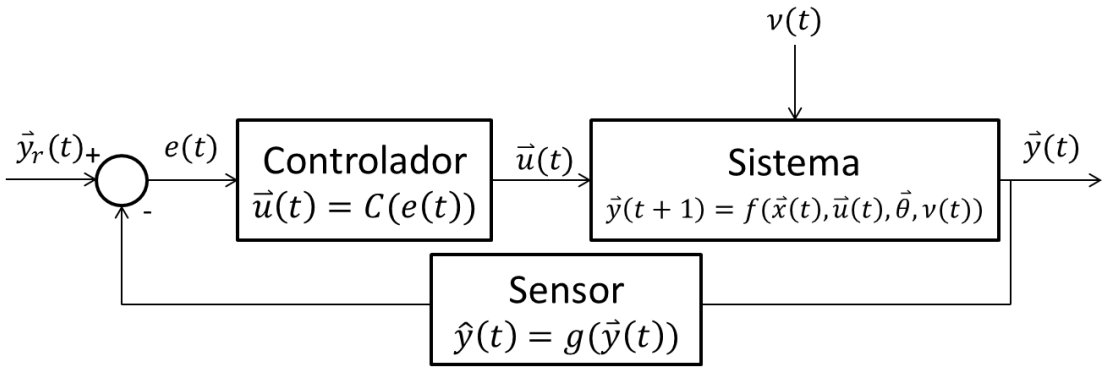


Figura 2.2: Esquema de control de lazo cerrado, incluyendo sus elementos más importantes: un sistema S sometido a perturbaciones $\boldsymbol{\nu(t)}$; un sensor que mide las salidas del sistema con un error asociado; y un controlador C que calcula la acción de control $\boldsymbol{u(t)}$ en base a las mediciones y a una referencia dada $\boldsymbol{y_r(t)}$.

Como se puede ver, el cálculo de la acción de control depende directamente de las mediciones obtenidas. Los sensores incluyen en su información de fábrica (*datasheets*) rangos de tolerancia para los errores en condiciones de operación dadas, lo que permite estimar los errores y determinar si se pueden considerar despreciables para cierta aplicación.

2.2. Componentes del Quad-Rotor

Actualmente existen en el mercado configuraciones estándares para la construcción y ensamblado de quad-rotors, las cuales contemplan los siguientes componentes [**2**]:

1. **La estructura (brazos y cuerpo central)**

Corresponde al cuerpo del quad-rotor, cuya función es servir de esqueleto para los componentes eléctricos y mecánicos de la nave. Dadas las dimensiones, se debe minimizar el peso de la estructura manteniendo una rigidez y resistencia que soporten y transmitan las fuerzas que actúan sobre la aeronave. Se suelen usar materiales como aluminio, fibra de vidrio y polímeros para estos componentes dada su alta resistencia y bajo peso (ver Figura 2.3), aunque también existen prototipos hechos de fibra de carbón [**3**].

1. **Motores**

Los motores más usados para UAVs de baja escala son eléctricos de tipo ***brushless DC*** (BLDC), los cuales no contemplan anillos rozantes para el cambio de polaridad en el rotor., sino que conmutan la polaridad en electroimanes ubicados en el estator. Estos motores funcionan de manera rápida y son alimentados por una fuente DC que es luego convertida en una señal AC de tres fases que se aplica a los electroimanes.

Figura 2.3: Ejemplo de estructura de un quad-rotor: brazos de aluminio; tren de aterrizaje y soporte central de fibra de vidrio [**4**]

Su rendimiento es superior a los motores DC clásicos, y además se puede alcanzar una respuesta más rápida, por lo que su uso en quad-rotors resulta muy práctico, siendo además muy durables [**5**]. Un esquema para un motor BLDC de dos polos se puede ver en la Figura 2.4, en donde el cambio en la polaridad de los imanes se conmuta de manera externa al variar las señales de entrada.

2. Hélices

Las hélices suelen fabricarse a partir de materiales compuestos como polímeros, los cuales poseen características anisotrópicas, por lo que resisten fuerzas muy altas en ciertas direcciones. Esto resulta particularmente útil para los quad-rotors ya que se sabe de antemano el tipo de fuerzas a las que estarán sometidas, entre las cuales se encuentran fuerzas centrífugas y de roce viscoso con el aire [**6**].

3. Controladores Electrónicos de Velocidad (*ESC*)

Estos componentes transforman una entrada lógica de tipo PWM en una señal AC de tres fases que alimenta los motores BLDC, siendo imprescindibles para su control preciso. Además, amplifican el voltaje de salida nominal de los controladores, los cuales trabajan usualmente en rangos de voltaje lógicos inferiores a los motores [**7**].

Para efectuar un correcto control de la velocidad del motor, se debe monitorear la posición del imán permanente ubicado en el rotor, con el fin de efectuar los cambios de polaridad necesarios en los electroimanes del estator. Las dos estrategias más usuales son: la medición de la fuerza electromotriz producida por el movimiento relativo de los imanes; y el uso de sensores magnéticos basados en el efecto Hall [**8**].

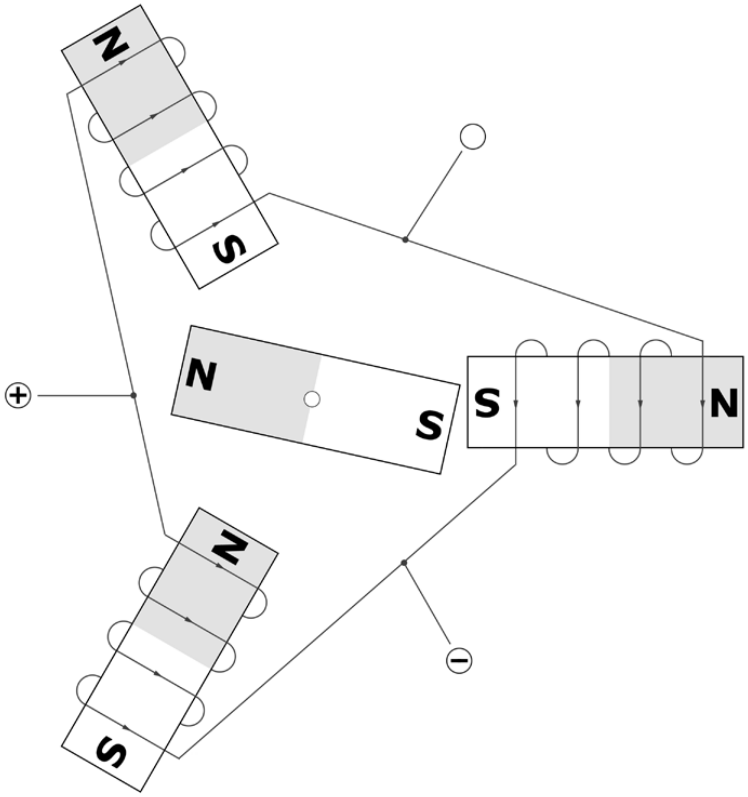


Figura 2.4: Esquema de un motor BLDC con dos polos. El imán permanente se ubica en el rotor mientras que los electroimanes se encuentran fijos en el estator, a diferencia de los motores con anillos rozantes, que ubican al electroimán en el rotor [**8**]

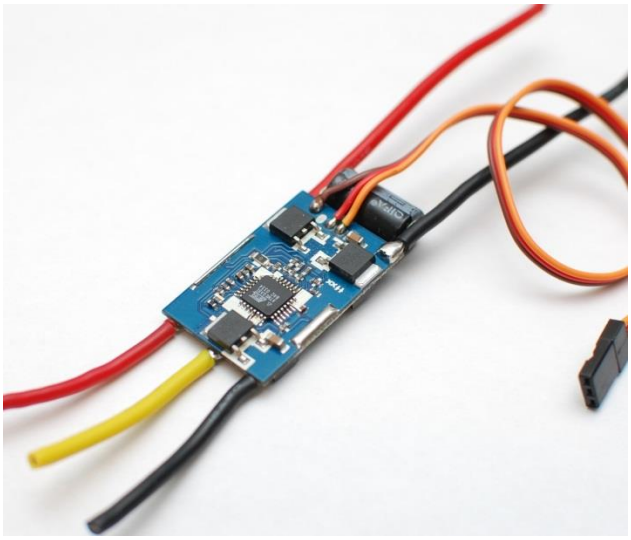


Figura 2.5: Controlador electrónico de velocidad. La señal PWM entra en voltajes menores a la alimentación DC, la cual es amplificada para salir como señal AC de 3 fases.

4. Batería

Se suelen utilizar baterías de polímero de litio, más comúnmente conocidas por la abreviación Li-Po, las cuales poseen una densidad de carga superior a las tradicionales ión-litio, además de un tamaño reducido y un rango de voltajes apropiado para aplicaciones de aeronaves radio-controladas. Las baterías Li-Po deben

ser utilizadas hasta un 80% de su corriente máxima. En [**9**] se realiza un cálculo del tiempo de vuelo máximo basado en la corriente que consume cada motor, obteniéndose una duración total de vuelo de 1h12m para motores BLDC suponiendo un consumo constante en torno a 2[A].

5. Controlador

Los controladores de postura, velocidad y aceleración actuales abundan y varían dependiendo del proveedor. Constan de un microprocesador y unidades de medición inercial. En general poseen puertos para diferentes protocolos como I2C, UART, SPI, USB, entre otros. Por otro lado, consideran salidas PWM (*pulse width modulation*) para manejar motores y otros periféricos, cuyo número varía desde 4 [**10**] hasta más de 12 [**11**].

Diferentes comunidades han colaborado a lo largo del tiempo para diseñar y optimizar diseños de controladores para su uso con *firmware* específico. Estos diseños se encuentran generalmente disponibles para su uso con distintos tipos de naves. Por otro lado, los proveedores comerciales de UAVs diseñan sus propios módulos de control. En general, el controlador de una nave *ready-to-fly* de esta índole tiene un costo menor al estar optimizado para ese vehículo en particular, en comparación a un módulo diseñado por comunidades *open-source*. Sin embargo, no permite muchas modificaciones y puede presentar una capacidad de procesamiento menor.

6. Sensores

En la Figura 2.2 se mostró que un esquema de control de lazo cerrado del quad-rotor necesita lecturas actualizadas de las coordenadas. Para ello, existe una gama de sensores que permiten obtener estos datos con diferentes niveles de precisión y rangos de operación.

Los sensores básicos utilizados en la integración de sistemas quad-rotor se reúnen en las Unidades de Medición Inercial (*IMU* por sus siglas en inglés), las cuales incluyen giroscopios y acelerómetros de 3 ejes de estado sólido en placas de tamaño conveniente [**12**]. Esto permite obtener, de manera rápida mediante integración, la orientación de la nave; es decir, una estimación de sus coordenadas $(\hat{\phi}, \hat{\theta}, \hat{\psi})$. Las unidades IMU se complementan con altímetros. Los más utilizados corresponden a sensores barométricos, que estiman la altitud basados en las medidas de presión, con un rango de operación que llega cerca de los 25000 [m].

Para el uso de quad-rotors, los sensores barométricos son ventajosos con su bajo costo, sin embargo en general las lecturas presentan imprecisiones importantes. Para ciertas aplicaciones esto puede ser poco apropiado, por lo que existen variantes más precisas como sensores basados en sonar, laser, u otros.

Por otro lado, el uso de un magnetómetro (o compás) permite tener una orientación con respecto a un sistema de referencia acoplado a la tierra, lo cual facilita la estimación de la pose con respecto a la referencia. Esto es complementado con el uso de GPS (*Global Positioning System*), el cual permite obtener la posición en hasta 3 ejes en tiempo real, dependiendo del número de satélites disponibles en el momento.

Otra estrategia usada en algunas líneas de investigación [**13**] contempla el uso de cámaras para estimar la posición y orientación del quad-rotor en ambientes controlados. Para ello, se usan marcas de colores en puntos específicos de la nave, las cuales son detectadas y con ello se calcula la posición y orientación. En primera instancia, los resultados indican que una cámara externa no es suficiente para realizar una buena estimación de las coordenadas del quad-rotor. En un trabajo posterior [**14**], se combinan 2 cámaras, una interna y una externa a la nave, para estimar su pose, obteniéndose resultados positivos.

Otros tipos de sensores existentes permiten realizar mediciones de diferentes variables. Los sensores de flujo óptico son utilizados para detectar puntos o marcadores que se utilizan para fines variados, desde detección de obstáculos hasta seguimiento de objetivos. Por otro lado, sensores laser también pueden ser

usados para detectar objetos y estimar su distancia a ellos. Esto puede resultar particularmente útil para realizar mapeo de espacios.

7. Radio Control

Para comunicar el quad-rotor con el operario se requiere un módulo de comunicación (transmisor-receptor) con un mínimo de 4 canales, uno por cada entrada. Operan en el rango de frecuencia de los 2.4 [GHz], y actualmente existe una amplia diversidad de módulos de comunicación, con modulación del tipo DSM, DSM2, FASST, ACSST, entre otros. Dependiendo del controlador utilizado, se debe utilizar algún tipo de *encoder* para transmitir más de un canal.

Por otro lado, se debe poseer un mando para actuar como interfaz con el usuario. Los mandos usuales tienen dos palancas o *sticks* de dos ejes cada uno. El stick izquierdo controla tanto la aceleración en el eje $\hat{z}$ de la nave (*throttle*) como la rotación del ángulo ψ (*yaw*), mientras que el derecho maneja la inclinación en los ángulos θ (*roll*) y ϕ (*pitch*). Además, en general existen otros canales disponibles para ser utilizados, por ejemplo, para cambiar modos de vuelo en línea. La Figura 2.6 muestra un ejemplo de mando de 6 canales con modos de vuelo y programación de distintos modelos para su uso con diferentes naves, entre otros.

Figura 2.6: Mando de control de 6 canales con opciones de programación.

8. Telemetría

Los módulos de telemetría permiten obtener datos de la nave en tiempo real en una estación en tierra. Operan en el rango de los 433 [MHz] y los 915 [MHz], dependiendo de la reglamentación local. La comunicación con el controlador se realiza a través del protocolo UART.

2.3. Proyectos de código abierto

Actualmente existen diversas comunidades en línea que trabajan activamente para desarrollar alternativas de código abierto (*open-source*) y gratuitas para el desarrollo de firmware y software con el objetivo de controlar UAVs de todo tipo: multi-rotores, helicópteros, aviones, entre otros. Además de proveer los códigos que se utilizan, se diseñan controladores específicos para este tipo de proyectos, los cuales integran diversos módulos (IMU, I/O, compás, entre otros). Dichos controladores, o más comúnmente

denominados *autopilots*, son posibles de adquirir a precios que varían dependiendo del fabricante. En esta sección se muestran algunas de las comunidades más prominentes en lo que respecta a drones. Una lista resumida de éstas se puede ver en la Tabla 2.2. Una versión más completa se encuentra disponible en [**15**].

Tabla 2.2: Resumen de comunidades open-source de UAVs más importantes

Nombre Proyecto	**Controlador Principal**	**Diseños hardware disponibles**	**Sensores considerados**	**Configuraciones**
AeroQuad	Arduino Pro Uno / Mini / Mega, STM32	Esquemáticos y PCBs	Gyro, Acc, GPS, Mag, Alt	Tri, Quad, Hexa, Octo
APM: Copter (ex-ArduPilot)	APM: Atmega2560 Pixhawk: STM32	Esquemáticos y PCBs	Gyro, Acc, GPS, Mag, Alt, Temp	Tri, Quad, Hexa, Octo
MultiWiiCopter	Arduino Pro Mini / Mega	Sólo esquemáticos	Gyro, Acc, Mag, Alt	Tri, Quad, Hexa
OpenPilot	STM32	Esquemáticos y PCBs	Gyro, Acc, GPS, Mag, Alt, Temp	Tri, Quad, Hexa, Octo, > Octo
UAVP (Universal Aerial Video Platform)	ATMEGA644	Solo esquemáticos	Gyro, Acc, GPS, Mag, Alt, Temp	Tri, Quad, Hexa, Octo, > Octo

En general, todas las comunidades presentan foros y guías dedicados tanto a instruir a los nuevos usuarios en la construcción e integración de aeronaves, típicamente quad-rotors. También poseen instructivos y manuales sobre diversos temas, listado de piezas, documentación sobre hardware y software, descarga de códigos (compilados y para su edición), además de enlaces a tiendas para adquirir componentes.

2.3.1. AEROQUAD

Las naves diseñadas por AeroQuad están basadas en procesadores Arduino (con procesadores ATMega2560), los cuales son muy populares debido a su bajo costo, aunque algunos controladores contemplan procesadores STM32. El software asociado, AeroQuad Flight Software, está diseñado para soportar ambos tipos de procesadores. Aunque no presenta un manual formal, existen guías que ayudan a la configuración y calibración de las naves. El firmware y software están enfocados en multi-cópteros, desde 3 hasta 8 hélices.

Los precios de los DIY kit ("hágalo-usted-mismo" - contienen todos los componentes electrónicos y mecánicos para ensamblar la nave) rondan entre los USD$420 y los USD$540, dependiendo de las opciones. Por otro lado, el precio del controlador es de USD$150.

2.3.2. ARDUPILOTMEGA

La comunidad de ArduPilotMega, abreviado APM, trabaja activamente en el desarrollo de vehículos aéreos y terrestres, teniendo 3 diferentes categorías: COPTER, para helicópteros y multi-cópteros; PLANE, para modelos de aviones; y ROVER, para vehículos terrestres. El hardware principal es idéntico: el controlador clásico APM se caracteriza por usar microprocesadores ATMega2560, inspirados por Arduino. Por otro lado, se desarrolla firmware diferenciado para cada tipo de configuración.

El software característico de esta comunidad es el APM: Mission Planner, cuya interfaz se presenta en la Figura 2.7. Este programa se caracteriza por poseer un manual detallado en línea, además de opciones de configuración avanzadas de alto nivel como *path-planning*. Por otro lado, permite la modificación de parámetros internos de bajo nivel, como las constantes de los controladores, de manera sencilla [**16**].

Figura 2.7: Interfaz gráfica del software Mission Planner, presentando un mapa para *path-planning* y mediciones en tiempo real de la nave

La comunidad se aloja en el sitio DIYDrones [**17**], mientras que la tienda asociada tiene como nombre 3DRobotics [**18**]. Los precios de los DIY kits rondan los USD$450 incluyendo módulos GPS, de telemetría y el controlador APM 2.6. Actualmente se está innovando con el nuevo controlador Pixhawk, el cual cambia a microprocesadores STM32, presentando un poder computacional más importante. Su precio es de USD$200, con opción de agregar módulos extra (GPS, telemetría, entre otros).

2.3.3. MULTIWIICOPTER

Este proyecto se inició con la idea de utilizar los sensores utilizados por los mandos de la consola Nintendo Wii, específicamente el WiiMotionPlus y el Nunchuk, los cuales poseen acelerómetros y giróscopos. Estos sensores se utilizan en conjunto con chips Arduino o similares. Si bien el desarrollo de software no ha sido del mismo nivel que otras alternativas, sí posee interfaces gráficas programadas en lenguaje Java que permiten configurar parámetros internos y visualizar opciones.

La comunidad se aloja en el sitio [**19**], sin embargo no poseen una tienda para compra directa de productos. Por otro lado, recomiendan sitios precisos para los componentes, además de proveer descarga de software y firmware propio para su utilización con multi-cópteros de hasta 6 rotores.

2.3.4. OPEN PILOT

La comunidad OpenPliot nació a fines del 2009 y actualmente ha desarrollado software y firmware capaces de volar multi-cópteros y aviones, con avances en navegación automática por GPS. Sin embargo, la mayoría del Hardware no se encuentra disponible comercialmente para el público ya que las pruebas no han sido completadas en su totalidad.

El sitio de la comunidad [**20**] aloja foros de discusión y permite la participación activa en el desarrollo de los algoritmos. Por otro lado, la tienda [**21**] ofrece la adquisición de algunos productos como el kit de hardware por USD$200 que incluye controlador, GPS, telemetría de 433MHz y antenas, pero sin posibilidad de adquirir el resto del equipamiento físico como *frame*, motores y ESCs, entre otros.

2.4. El control de ArduPilotMega

2.4.1. MODOS DE VUELO DEL *FIRMWARE* APM: COPTER

En general, cada *firmware* diferente implementa lazos de control de manera independiente, los cuales varían dependiendo del modo de vuelo que se utilice. A continuación se detallan algunos modos de vuelo existentes en el *firmware* APM: Copter, especificando las variables que se controlan con las palancas, o *sticks*, del mando.

1. *Stabilize*

El modo de vuelo estabilizador es el más simple en términos de maniobrabilidad, ya que los ángulos de inclinación en torno a los ejes en el plano horizontal, θ y ϕ, son manejados directamente con el *stick* derecho; al ser soltados la nave vuelve los ángulos a 0°, nivelada con respecto al suelo. El stick izquierdo maneja por un lado la velocidad de los motores, y por lo tanto la aceleración $\ddot{z}$, y por otro lado la tasa de cambio del ángulo de dirección, $\dot{\psi}$.

2. *Alt-Hold*

El modo de retención de altitud o *alt-hold* es similar al modo estabilizador, con la diferencia que la altitud es mantenida al variar los valores de los ángulos θ, ϕ y al desplazar la nave en el plano horizontal. El eje horizontal del *stick* izquierdo maneja la tasa de cambio del ángulo de dirección, $\dot{\psi}$, mientras que el eje vertical indica la velocidad de ascenso o descenso, $\dot{z}$, con respecto al eje vertical de referencia. Cabe destacar que la retención de altitud se realiza en base a las mediciones obtenidas a través de los sensores instalados que, como se indicó en la Sección 2.2, pueden ser erráticas en ciertas condiciones atmosféricas o ambientales, lo que llevaría a tener desplazamientos indeseados en el eje z.

3. *Loiter*

En el modo *loiter*, la nave mantiene su posición y orientación actuales. El *stick* derecho indica la velocidad en los ejes x e y, mientras que el *stick* izquierdo controla, al igual que en el modo *Alt-Hold*, la aceleración vertical $\dot{z}$ y la rotación $\dot{\psi}$. Este modo requiere una buena señal GPS, lo que limita su utilización a aplicaciones al aire libre.

4. *Auto*

En este modo, la nave sigue un plan de vuelo pre-programado el cual consiste en puntos de pasada o *waypoints*, indicando los lugares a los cuales se dirigirá en orden el UAV, y órdenes de otros tipos como esperar, girar, guardar información, manejar *servos*, entre otros. Al igual que el modo *loiter*, en Auto el UAV necesita tener buena señal GPS para poder operar correctamente, ya que los *waypoints* se definen en un mapa en el software Mission Planner.

Tabla 2.3: Resumen de controles del mando para diferentes modos de vuelo del firmware APM: Copter

Modo de Vuelo	Stick Izquierdo		Stick Derecho		Necesita GPS
	Eje |	Eje -	Eje |	Eje -	
Stabilize	F_z	$\dot{\psi}$	ϕ	θ	No
Alt-hold	$\dot{z}$	$\dot{\psi}$	ϕ	θ	No
Loiter	$\dot{z}$	$\dot{\psi}$	$\dot{y}$	$\dot{x}$	Si
Auto	-	$\dot{\psi}$	-	-	Si
Acro	F_z	$\dot{\psi}$	$\dot{\phi}$	$\dot{\theta}$	No
Drift	F_z	-	ϕ	ψ (θ)	No

En la Tabla 2.3 se resumen los diferentes controles en los modos de vuelo descritos, además de algunos otros que no fueron mencionados [**22**].

2.4.2. LAZO DE CONTROL DEL MODO ESTABILIZADOR

Las Figuras 2.8 y 2.9 detallan el lazo de control del modo estabilizador de manera resumida [**23**].

El control de la postura en modo estabilizador se maneja mediante la inclusión de 4 ganancias por ángulo (+1 para el ángulo de dirección ψ). La ganancia "K_P STAB" regula el controlador P del ángulo correspondiente, mientras que las constantes K_P, K_I, K_D regulan el controlador PID asociado a la tasa de cambio de cada ángulo. Entre ambos existe una traslación de la referencia externa F_0 a la referencia propia de la nave, la cual se define por el vector de traslación $\overrightarrow{x_t}$. Por otro lado, existe un término estabilizador de la entrada en el ángulo de dirección. Dicho término tiene como objetivo atenuar los efectos indeseados en esa orientación debidos a diversos motivos como vibraciones y efectos giroscópicos, entre otros.

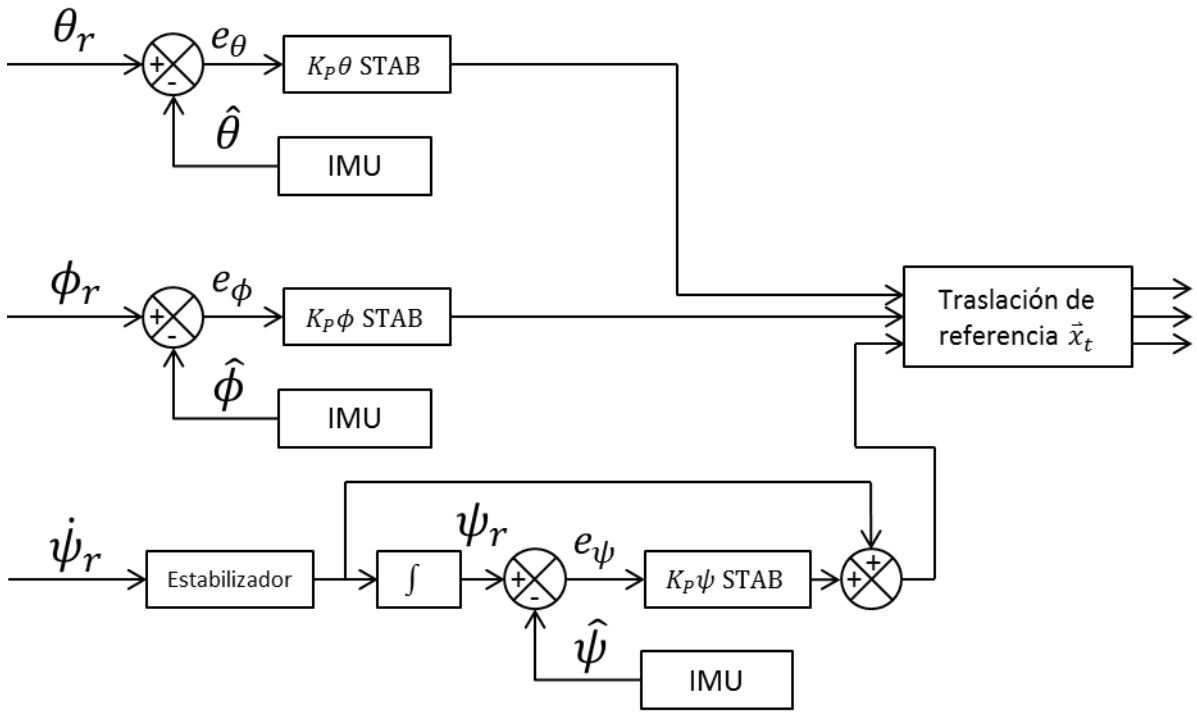


Figura 2.8: Control de los ángulos en modo estabilizador basado en los comandos entregados por el usuario mediante el mando RC. Corresponde al control de alto nivel, y entrega tasas de cambio de los ángulos a los controladores de bajo nivel.

En general existen dos conjuntos de variables de las que depende la maniobrabilidad. El primero corresponde a las ganancias de los controladores P/PID que deben ser sintonizadas dependiendo de la nave. El segundo conjunto no se ve explícitamente en los diagramas de las Figuras anteriores, y consiste en las constantes de calibración del mando RC que regulan su sensibilidad. Las ganancias K_P, K_I, K_D no deberían ser modificadas una vez ajustadas ya que sus valores influyen tanto en el seguimiento de referencia impuesta por el usuario así como la estabilización misma del modo estabilizador.

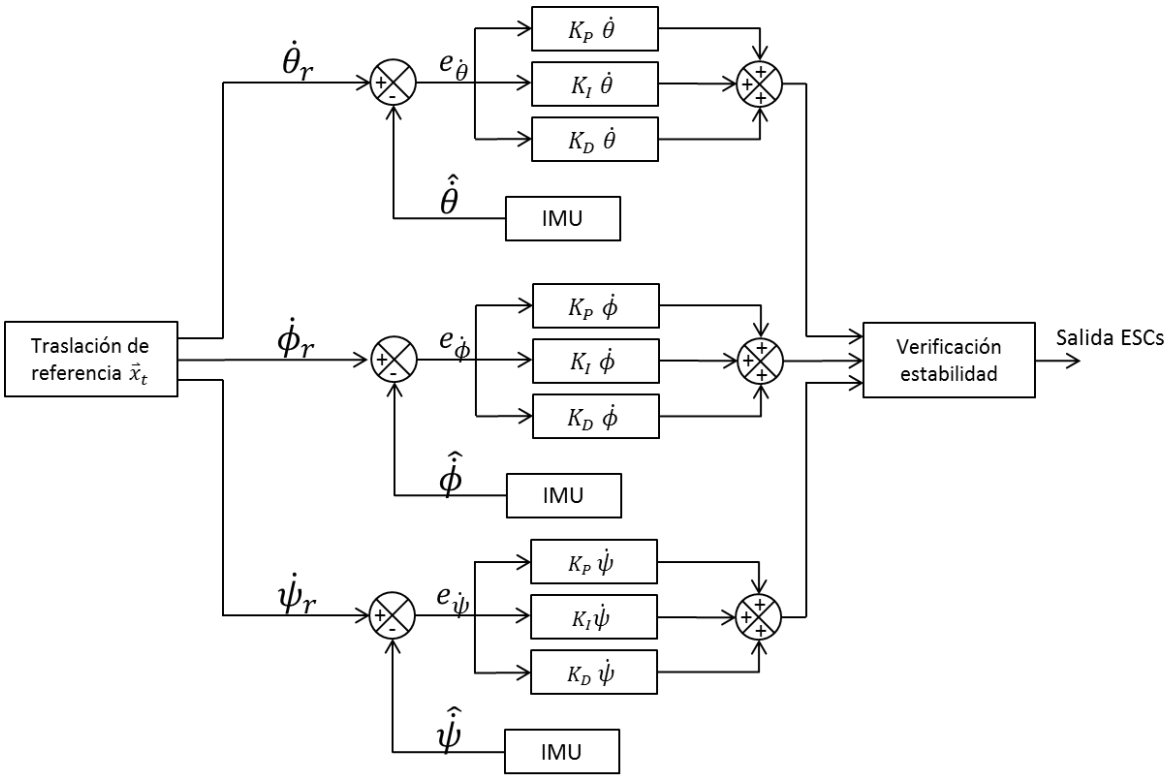


Figura 2.9: Control de la tasa de cambio de los ángulos en modo estabilizador basado en las referencias entregadas por el controlador de ángulos. Corresponde al control de bajo nivel, y entrega la frecuencia de giro de los motores en señales PWM.

2.5. Autonomía y toma de decisiones

Un tema de creciente importancia involucrando a los vehículos aéreos no tripulados corresponde a la toma de decisiones que debe realizar un sistema controlador para poder volar de manera autónoma. Para ello, las acciones deben realizarse considerando tanto el entorno como los objetivos de la misión en particular.

En la actualidad, se desarrollan tanto estrategias de toma de decisiones centralizadas como algoritmos descentralizados [**24**]. El primer enfoque tiene la ventaja de asignar tareas sin conflictos existentes, ya que todo el procesamiento de alto nivel es realizado por una unidad central, lo que es ideal para aplicaciones en donde participa una sola nave. Sin embargo, requiere una conexión estable a una estación fija que procesa la información para asegurar un conocimiento situacional consistente. Esto se traduce en la constante atención a cambios en el entorno, específicamente objetos, clima, así como en cambios en la misión en particular.

Los algoritmos descentralizados son más ventajosos para aplicaciones en donde grupos de UAVs trabajan en conjunto ya que reducen los tiempos de comunicación y aumentan la robustez del sistema frente a cambios. Estas estrategias se basan en la comunicación periódica entre naves que computan sus propias tareas, lo que puede llevar a tener conflictos en las decisiones. Es por ello que el sistema debe ser diseñado teniendo en mente estos aspectos.

Estimación del estado de carga de una batería

En cualquier tipo de estrategia una variable limitante es la energía disponible, lo que se traduce en tiempo de vuelo. Este concepto se conoce como la autonomía del sistema, la cual está íntimamente ligada al tiempo de descarga (RUL – *remaining-useful-life*). El estado de carga (SOC - *state-of-charge*) de las baterías utilizadas es por lo tanto una variable que debe ser monitoreada constantemente para poder determinar la energía disponible. Sin embargo, este valor no es medible directamente, por lo que se

presenta un problema de observabilidad en donde el SOC debe ser estimado en base a las variables que se pueden medir, como el voltaje instantáneo, la corriente consumida y la temperatura.

Para dar cuenta del tiempo de descarga, se debe realizar un pronóstico en base al perfil de utilización y el SOC, entre otros. En la actualidad existen diversos modelos de baterías, los cuales se han desarrollado bajo distintos paradigmas, como circuitos equivalentes y modelación electroquímica [**25**]. Estos enfoques buscan, en última instancia, encontrar modelos del SOC de las baterías. Sin embargo, en lugar de realizar un modelamiento exhaustivo, se puede trabajar utilizando un enfoque bayesiano, en donde se combinan los conocimientos *a priori* de los modelos con un aprendizaje basado en muestras obtenidas a partir de experimentación. El enfoque más utilizado para ello es el filtro de inferencia bayesiana, o también llamado filtro de partículas (PF) [**26**], el cual permite realizar un pronóstico de un intervalo de confianza del tiempo de descarga. Esto puede utilizarse para coordinar misiones y asignar distintas instrucciones en línea, lo que ayuda a optimizar el uso de energía de uno o más UAVs trabajando con objetivos específicos.

Capítulo 3: Desarrollo de un asistente de vuelo

Este capítulo detalla el desarrollo de las actividades del trabajo, con énfasis en la metodología y los procedimientos realizados. En primer lugar, en la Sección 3.1 se muestran los componentes a utilizar y se detalla el proceso de integración de los elementos electrónicos y mecánicos que tuvo lugar. Dicha sección además contempla problemas encontrados en el momento de la puesta en marcha y las soluciones aportadas, así como los ajustes a los parámetros y la sintonización de controladores. Posteriormente, en la Sección 3.2 se presenta el proceso de sintonización de parámetros de los controladores, así como las pruebas preliminares que se realizaron para verificar el correcto funcionamiento de todos los módulos. Finalmente, en la Sección 3.3 se explica el diseño del asistente de vuelo, comenzando por una descripción del problema general y la solución propuesta, para luego describir en detalle la implementación de cada uno de los módulos de dicho sistema.

3.1. Integración de la plataforma de vuelo

Los elementos necesarios para la integración de un UAV se detallaron en la Sección 2.2. En la presente Sección se aborda el proceso de montaje de la plataforma, destacando los problemas encontrados y las soluciones aportadas.

En primer lugar, se debe destacar que, dada la experiencia del equipo del Laboratorio de Campo del AMTC con UAVs, se utilizó la opción APM: Copter para el *hardware*, *software* y *firmware*.

3.1.1. COMPONENTES DISPONIBLES

Al momento de comenzar el trabajo, en el Laboratorio de Campo se encontraban componentes a disposición que serían utilizados para la integración del quad-rotor. Dichos elementos se listan a continuación:

- Frame: Q450 450mm. Brazos de nylon poliamida, centro de fibra de vidrio (Figura 3.1)

Figura 3.1: Frame Q450 utilizado

- Motores: 4x EMAX CF2822, BLDC, 7000 RPM, 14.5[A].

Figura 3.2: Motor EMAX CF2822

- ESCs: 4x HK-SS30A. Corriente maxima: 25 [A].
- Módulo GPS: CN06 Plus, incluye compás. Conexiones DF13 de 4 pines.
- RC: Optic 6 Sport 2.4, con módulos FrSky D8RII – Plus, modulación ACCST.
- Telemetría: 2 módulos RCTimer Radio Telemetry 433MHz con FHSS.
- Baterías: Turnigy Nano-Tech 5000mAh 35-70C 3s 11.1V.
- Cables: AWG12 para alimentación a los ESC.
- Otros: Conectores tipo *bullet* y XT-60, protección termo retráctil, correas con velcro, soldadura y herramientas.

3.1.2. CONTROLADOR

El módulo principal del quad-rotor debió ser encargado en la etapa de planificación del trabajo. Las opciones de controladores compatibles con el *firmware* APM: Copter eran las siguientes [**18**]:

- El controlador clásico **APM 2.6**, basado en un chip ATMega2560, posee el poder computacional de un Arduino.
- El controlador de transición **PX4**, basado en un chip más avanzado STM32, fue diseñado de manera de comenzar la implementación de un nuevo tipo de controladores. Tiene un número de módulos de expansión para diversificar su uso.
- El nuevo controlador **Pixhawk**, que representa la modernización del PX4. El procesador está basado en un potente chip STM32, y con 14 salidas PWM/Servo. Posee además puertos de diversos protocolos (I2C, UART, CAN) para conectar variados periféricos.

La Tabla 3.1 resume las características más importantes de los tres controladores.

Las 3 opciones han sido probadas en muchas ocasiones por la comunidad, mostrando capacidad de vuelo estable en todos los casos. Por otro lado, la posibilidad de integrar periféricos y módulos adicionales, como cámaras o sensores, además de la velocidad de procesamiento y tamaño de memoria, hacen muy atractivo el Pixhawk, por lo que se optó por esta solución con el fin de tener opciones de ejecutar programas de más alto nivel en el futuro, y de no tener restricciones computacionales en el mediano plazo. La Figura 3.3 muestra una imagen del controlador escogido.

Tabla 3.1: Características de los controladores compatibles con APM: Copter

Controlador	Procesador	Sensores	PWM Out	Interfaces
APM 2.6	ATMega2560 16MHz / 8KB RAM / 256KB Flash	Gyro, Acc, Mag, Alt. Adicional: Gyro+Acc (6 ejes)	14	I2C, UART, micro USB, PPM-Sum
PX4	STM32 Cortex-F4 168MHz / 192KB RAM / 1MB Flash	Gyro, Acc, Mag, Alt.	4	3x UART, PPM-Sum, I2C, SPI. Expandibles.
Pixhawk	STM32 Cortex-F4 168MHz / 256KB RAM / 2MB Flash	Gyro, Acc, Mag, Alt. Adicional: Gyro+Acc (6 ejes)	14	5x UART, 2x CAN, I2C, SPI, micro USB, Spek. DSM, PPM-Sum

Figura 3.3: Controlador Pixhawk, de 3DRobotics, compatible con el firmware APM: Copter

Los módulos de comunicación RC de 2.4MHz pueden presentar como salidas dos tipos de señales: una por cada canal del tipo PWM, o bien una señal codificada del tipo PPM que equivale al *encoding* de las señales de los diferentes canales presentadas en una sola salida. El módulo de control Pixhawk, por otro lado, posee una interfaz de entrada del tipo PPM, por lo que para poder compatibilizar la telemetría PWM se debe integrar un módulo codificador, o "Encoder", el cual fue pedido en conjunto con el controlador.

3.1.3. MONTAJE DE LA ESTRUCTURA Y DISPOSICIÓN DE COMPONENTES

La integración de un UAV contempló varios pasos a seguir. En primer lugar, se realizó un diseño de la disposición general de los componentes, la cual consideró diversos factores como minimización del cableado y de interferencia electromagnética entre componentes, así como el equilibrio de la masa total de la nave. A continuación, se procedió a ensamblar la estructura (*frame*) y disponer los componentes de manera segura y firme. Las conexiones se realizaron teniendo el cuidado de que todos los cables se mantuvieran firmes aún bajo grandes aceleraciones.

Para la disposición de componentes se tuvieron las siguientes precauciones adicionales:

- Las especificaciones de los motores indican que a 7000RPM consumen 14.5 [A] cada uno. Esto puede generar campos electromagnéticos no despreciables, lo cual puede causar interferencia tanto en los sensores primarios del quad-rotor (compás, acelerómetro, giroscopio) así como en el circuito del controlador mismo. Por tanto, se debe maximizar la distancia entre la batería y sus cables de alimentación hacia los ESC, y los módulos Pixhawk y GPS.

- El módulo GPS requiere una conexión directa con los satélites para funcionar. Esto se logra posicionándolo en la parte superior de la nave.
- Los sensores de la unidad IMU son sensibles frente a perturbaciones como vibraciones. Para reducir sus efectos negativos en el desempeño de la nave se debe utilizar algún material que disipe las vibraciones entre el controlador y el *frame*.

Con estos factores en cuenta, se logró en primer lugar ensamblar la estructura basal para luego disponer todos los componentes de manera adecuada. Se encontraron las siguientes dificultades al momento de posicionar algunos elementos:

1. Reemplazo de conectores '*bullet'*: Los terminales de tipo *bullet* presentes en los motores y los ESCs se encontraban en malas condiciones, por lo que fueron repuestos, procurando protegerlos posteriormente con material termo-retráctil para evitar cortocircuitos.
2. Alimentación desde la batería: Se reemplazó el terminal del módulo de distribución de poder para concordar con el conector *bullet* de la batería. Además, se construyó un arnés para alimentar los 4 ESCs desde el módulo de alimentación.
3. Ranuras de fijación de motores: Los motores disponibles presentaban ranuras de 3mm de diámetro a 11mm de radio. Sin embargo, el *frame* disponía de ranuras a menor distancia radial (8.75mm), cono se muestra en la Figura 3.4. Se decidió utilizar ranuras de mayor diámetro ubicadas a 11mm del centro en conjunto con placas auxiliares para poder fijar los pernos desde el inferior. La Figura 3.5 muestra un corte transversal de la estructura con la solución indicada.

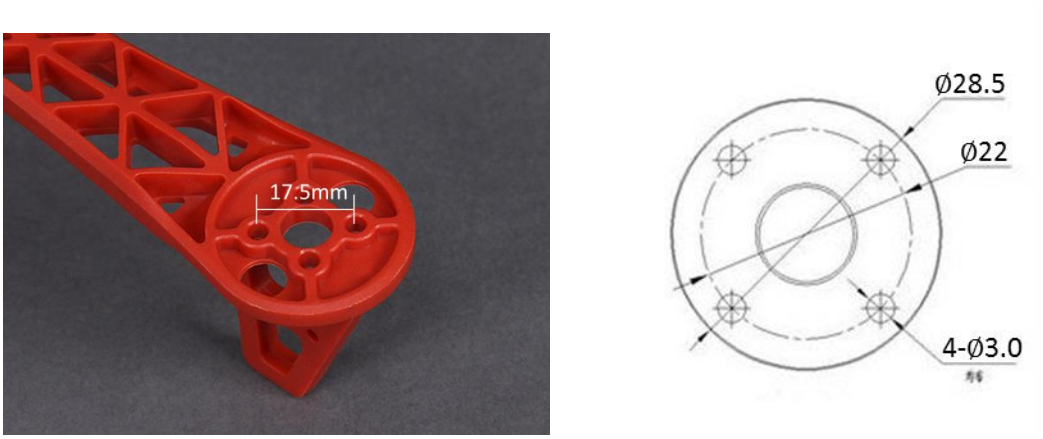


Figura 3.4: Medidas de las ranuras ubicadas en el frame (izq.) y en los motores (der.). Las ranuras de 3 [mm] ubicadas en el *frame* no coincidían con las ranuras de los motores. Las ranuras más grandes de 8 [mm] se encontraban a una distancia donde coincidían con los agujeros de los motores.

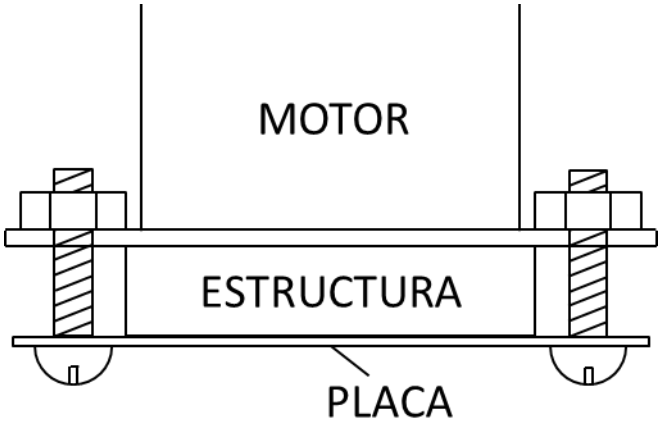


Figura 3.5: Diagrama que muestra la fijación de motores utilizando una placa metálica auxiliar

4. Posicionamiento del módulo GPS: Para ubicar este componente en la parte superior se diseñó una placa auxiliar para elevarlo sobre el controlador. Dicha placa fue fijada mediante separadores de nylon con hilo interior, los cuales se ubicaron en ranuras de la placa superior del *frame*. Se ubicó el módulo GPS sobre dicha placa como se muestra en la Figura 3.6.

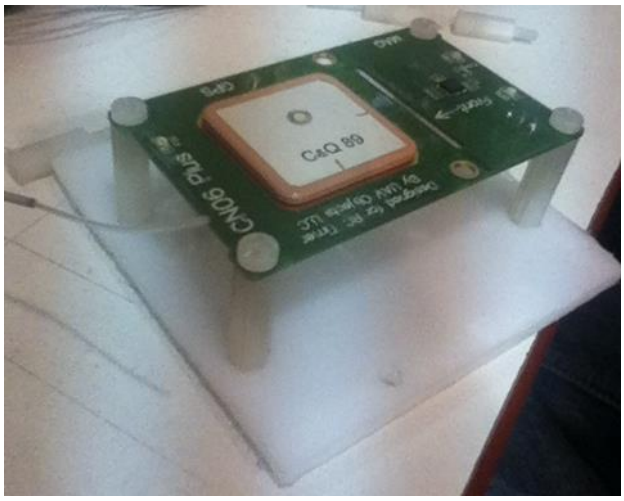


Figura 3.6: Módulo GPS montado sobre la placa auxiliar utilizando separadores de nylon

5. Reemplazo de terminales DF13: Las conexiones entre el controlador y periféricos se realizan en general utilizando terminales de 2 a 6 pines. Para conectar el compás externo, se debió reemplazar un terminal de 6 pines por uno de 4 pines, teniendo cuidado de mantener su integridad.

3.1.4. INSTALACIÓN DE SOFTWARE Y FIRMWARE

Para la instalación del *firmware* a utilizar, el **APM: Copter**, en primer lugar se instaló el *software* Mission Planner en un PC. Este fue descargado del sitio oficial de APM bajo la licencia pública general de GNU para programas de código abierto [**27**]. Al momento de la realización de este trabajo se utilizó la versión 1.3.5.

Una vez instalado, se conectó el controlador Pixhawk al PC a través de un cable microUSB. En primer lugar el *software* detectó el dispositivo y procedió con la descarga e instalación del *firmware* APM: Copter en su última versión estable 3.1.5. Se asignó la configuración del quad-rotor, siendo para esta nave la disposición en 'X'. La Figura 3.7 muestra la diferencia entre las configuraciones 'X' y '+'.

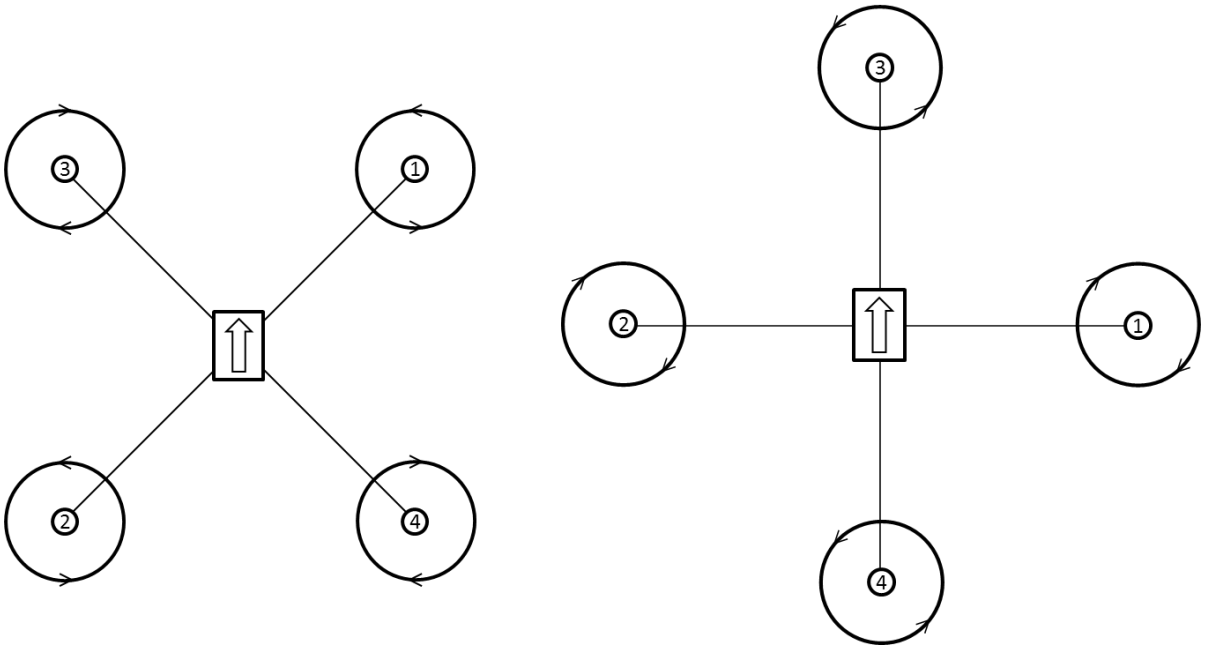


Figura 3.7: Configuraciones del quad-rotor: 'X' (izquierda) y '+' (derecha). La disposición de componentes debe ser acorde a la orientación deseada debido a las mediciones que se obtienen de los instrumentos (IMU y compás).

A continuación se procedió a realizar las calibraciones de los módulos necesarios para las mediciones de postura, para luego calibrar el mando de radiocontrol. Esto se realizó a través del *software* Mission Planner. Una vez completadas estas etapas, se debieron sintonizar los parámetros de los controladores de bajo y alto nivel del modo básico estabilizador.

3.2. Pruebas básicas de funcionamiento

3.2.1. SINTONIZACIÓN DE PARÁMETROS

El firmware APM: Copter incluye valores predeterminados para todos sus parámetros, los cuales determinan tanto las opciones de uso del *firmware* así como las constantes que se utilizan en los controladores de los distintos modos de vuelo vistos en la Sección 2.4. Los primeros parámetros pueden ser configurados manualmente. Sin embargo, las ganancias de los controladores deben ser ajustadas mediante un proceso iterativo de sintonización.

Los parámetros se pueden modificar en un menú de configuración del software Mission Planner como se ve en la Figura 3.8.

Para realizar la sintonización se consideraron los parámetros que tendrían influencia en el comportamiento del controlador del modo estabilizador. Estos valores corresponden a las ganancias P, I, D de la tasa de cambio de los ángulos (denotados K_P, K_I, K_D para los ángulos $\dot{\theta}, \dot{\phi}, \dot{\psi}$ en la 2.9) y las ganancias P del control de los ángulos (denotados K_P para los ángulos θ, ϕ, ψ en la 2.8). Si bien en la interfaz de configuración en Mission Planner existen más parámetros, éstos afectan el comportamiento de otros controladores y modalidades del *firmware*.

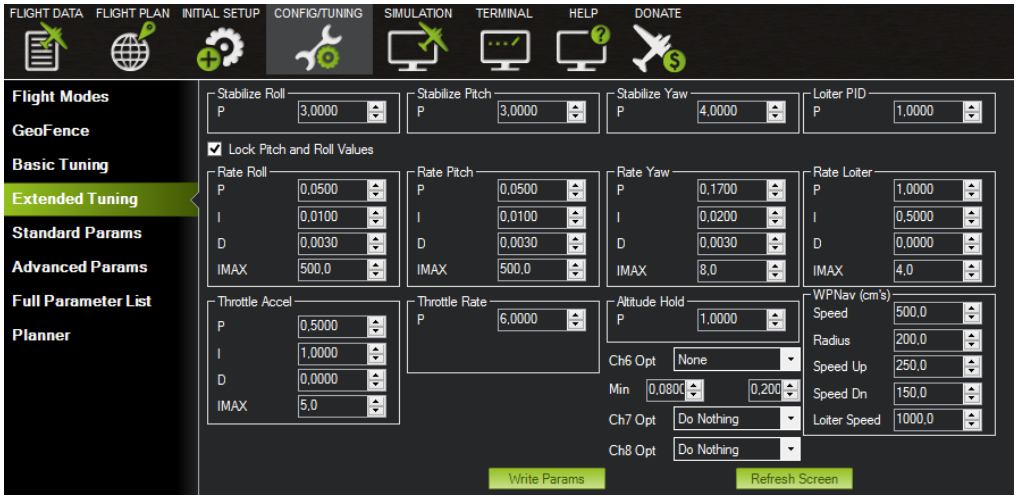


Figura 3.8: Interfaz de modificación de parámetros de los controladores

En un principio, los valores utilizados fueron los que se muestran en la Tabla 3.2, siguiendo las recomendaciones de sintonización para el modo estabilizador [**28**] [**29**].

Tabla 3.2: Valores iniciales de las ganancias de los controladores

K_P	θ	1
	ϕ	
	ψ	
K_P	$\dot{\theta}$	0.01
	$\dot{\phi}$	
	$\dot{\psi}$	
K_I	$\dot{\theta}$	0
	$\dot{\phi}$	
	$\dot{\psi}$	
K_D	$\dot{\theta}$	0
	$\dot{\phi}$	
	$\dot{\psi}$	

En una primera prueba, se logró elevar la nave y maniobrarla de manera básica. Sin embargo, las respuestas de la nave fueron lentas. Esto obligó a elevar los valores de los K_p a 3 y 0.1 respectivamente. De esta manera se logró una respuesta más rápida y sin oscilaciones importantes.

El siguiente paso fue configurar el parámetro K_D. Para ello, se asignó un valor inicial de 0.001 y se fue elevando en pruebas sucesivas, hasta llegar a un valor de 0.003. Con este valor se observó una mejoría en la respuesta de los controladores, disminuyendo las oscilaciones en la respuesta transitoria al momento de realizar un cambio importante de referencia. En este proceso se observó que al aumentar el valor de K_D para el ángulo de dirección ψ, la respuesta se volvió más lenta por lo que se modificaron los valores de K_P a 4 y 0.17, los cuales permitieron una respuesta más rápida y sin llegar a oscilar.

Finalmente, las ganancias K_I de todos los ángulos fueron inicializadas en 0.001. En principio no se mostró un cambio en la respuesta, sin embargo al aumentar dichos valores se observó una disminución en el tiempo de respuesta. Al superar el valor de 0.01 en los ángulos $\dot{\theta}$ y $\dot{\phi}$ se empezó a ver un sobrepaso importante al momento de ordenar cambios de referencia, lo que a su vez implicó la aparición de

oscilaciones. Por otro lado, para el ángulo de dirección $\dot{\psi}$ esto no ocurrió sino hasta tener un valor de $K_I = 0.02$.

Tabla 3.3: Valores finales de las ganancias de los controladores

K_P	θ	3.0
	ϕ	
	ψ	4.0
K_P	$\dot{\theta}$	0.5
	$\dot{\phi}$	
	$\dot{\psi}$	0.17
K_I	$\dot{\theta}$	0.001
	$\dot{\phi}$	
	$\dot{\psi}$	0.02
K_D	$\dot{\theta}$	0.003
	$\dot{\phi}$	
	$\dot{\psi}$	

Con esto, los valores finales obtenidos luego del proceso de sintonización se encuentran resumidos en la

Tabla 3.3. Se destaca que para los ángulos θ y ϕ se utilizan los mismos parámetros de manera automática debido a la simetría de la nave.

3.2.2. PRUEBAS BÁSICAS DE DESEMPEÑO

Una vez finalizada la sintonización de los parámetros básicos para el control de la nave, se realizaron algunas pruebas básicas de vuelo para evaluar el rendimiento general [**30**]. Dichas pruebas fueron realizadas por Rodrigo Asenjo, Ingeniero de Desarrollo en el Laboratorio de Robótica de Campo, quien cuenta con vasta experiencia de manejo de drones (quad-copters, hexa-copters, planeadores, entre otros).

Los gráficos se obtuvieron a través del software Mission Planner, gracias a su herramienta de análisis de registros de vuelo.

Seguimiento de referencia

En primer lugar, se muestran en el Anexo XI (pág. 82) los comandos de ángulos de referencia deseados en conjunto con los ángulos reales. Todas las unidades se encuentran en grados.

Se puede ver que, en general, la pose de la nave sigue a las referencias con un desfase de aproximadamente 0,1 [s], lo que se considera aceptable. Se observa además el acoplamiento entre las variables a través de los cambios en diferentes ángulos de manera simultánea. Un ejemplo de este comportamiento se presenta en la Figura 3.9.

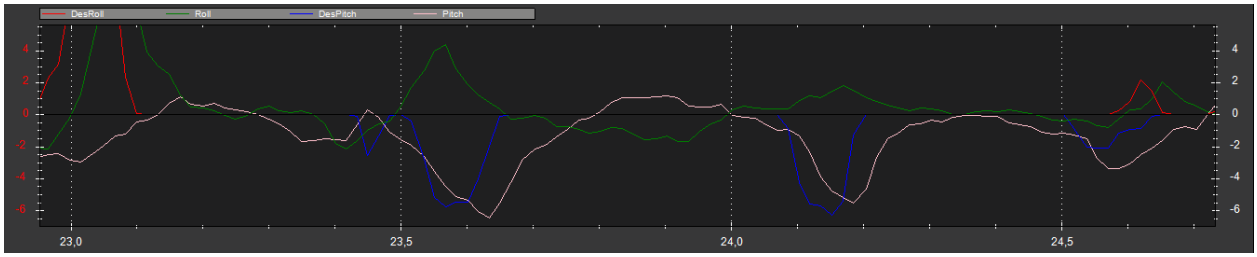


Figura 3.9: Angulos *roll* (verde) y *pitch* (rosa), en conjunto con las referencias pedidas (rojo y azul respectivamente): se observa el acoplamiento entre variables al pedir un cambio en el ángulo de elevación, y observándose un aumento en el ángulo de alabeo (~23.5 en el eje *x*), aun cuando su referencia se encuentra en 0.

Este acoplamiento, en conjunto con las no-linealidades del sistema, explican los comportamientos de las variables al estar usando controladores del tipo PID. Además, la sintonización se realiza a modo de prueba y error, lo cual representa un método cualitativo de determinación de parámetros. Esta alternativa solo es posible debido a la experiencia del personal del laboratorio y a la vasta información al respecto disponible en la comunidad de APM: Copter.

Vibraciones

Se debe determinar la precisión de los sensores inerciales de la nave, lo cual se realiza a través de un análisis de las mediciones obtenidas por los acelerómetros. Un gráfico para cada medición en la prueba completa se encuentra en el Anexo XII (pág. 84), con unidades de $\frac{m}{s^2}$.

Para determinar los límites de las vibraciones, se siguen las recomendaciones de la comunidad de APM: Copter [**30**]. Se establece que los límites para los ejes X e Y sean de $\pm 3\frac{m}{s^2}$, y en el eje Z de $\pm 5\frac{m}{s^2}$ en estado de reposo durante el vuelo (posición estabilizada).

A partir de los gráficos se observa que las mediciones transgreden dichos límites en varias ocasiones. Sin embargo, se entiende que además la nave se encuentra realizando movimientos. Para que el análisis sea más preciso, se procede a extraer un extracto más estable del vuelo, en donde se producen mayoritariamente desplazamientos en el eje Z, lo que se ilustra en la Figura 3.10. Se puede ver que en efecto las vibraciones en los ejes X e Y se limitan entre aproximadamente $\pm 3\frac{m}{s^2}$, mientras que en el eje Z se observan fluctuaciones, debidas a los desplazamientos verticales. Las vibraciones, por otro lado, permanecen despreciables en comparación a dichas fluctuaciones.

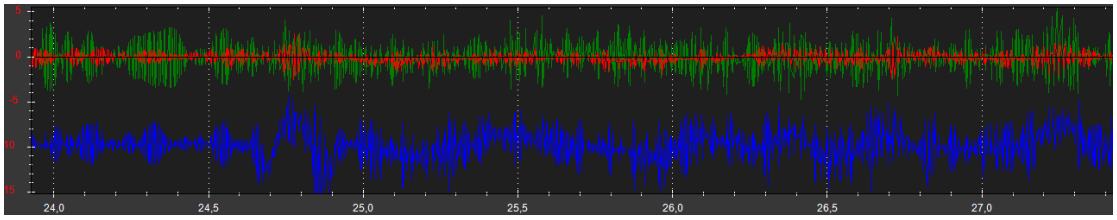


Figura 3.10: Mediciones del acelerómetro en los 3 ejes. Se aisló una porción de la prueba en la que la nave se mantuvo sin desplazamientos mayores en los ejes X e Y para analizar las vibraciones.

Corriente

Se visualiza la corriente circulante en el Anexo XIII (pág. 85). Se puede observar que se llega a un máximo de ~50A, lo que cae dentro del margen de seguridad del cableado utilizado. Considerando que en el vuelo de prueba se mantuvo una postura relativamente estabilizada, se puede concluir que estos son valores medios en el uso normal de la nave, por lo que no existe riesgo de sobrecargar los cables de alimentación.

Interferencia electromagnética

Un módulo importante del quad-rotor es el compás, o magnetómetro. Éste debe estar posicionado de tal manera que se minimice la interferencia creada por campos electromagnéticos generados por las altas corrientes que circulan de la batería a los motores.

En este caso, el compás se encuentra unido físicamente al módulo GPS, el cual se ubica en la parte superior de la nave, sobre la placa auxiliar. Esta placa fue diseñada específicamente con el propósito de alejar ambos elementos de los cables de alimentación. Para verificar la interferencia, a partir de los registros de telemetría (`.tlog`) se obtuvieron gráficos del campo "`mag_field`", el cual corresponde a la norma Euclideana del vector (mag_X, mag_Y, mag_Z):

$$mag\_field = \sqrt{(mag_X)^2 + (mag_Y)^2 + (mag_Z)^2}$$

Este valor se grafica en conjunto con la aceleración pedida por el operario en unidades porcentuales. Nuevamente, como indican las recomendaciones generales [**30**], una fluctuación porcentual de hasta 60% del campo magnético se considera aceptable. En el Anexo XIV se encuentra el gráfico de la prueba completa.

Se puede ver que en general el campo magnético fluctúa entre 200 y 240, lo que representa una variación del 10%. Sin embargo, se observa un descenso a ~140, lo que se ilustra de mejor manera en la Figura 3.11.

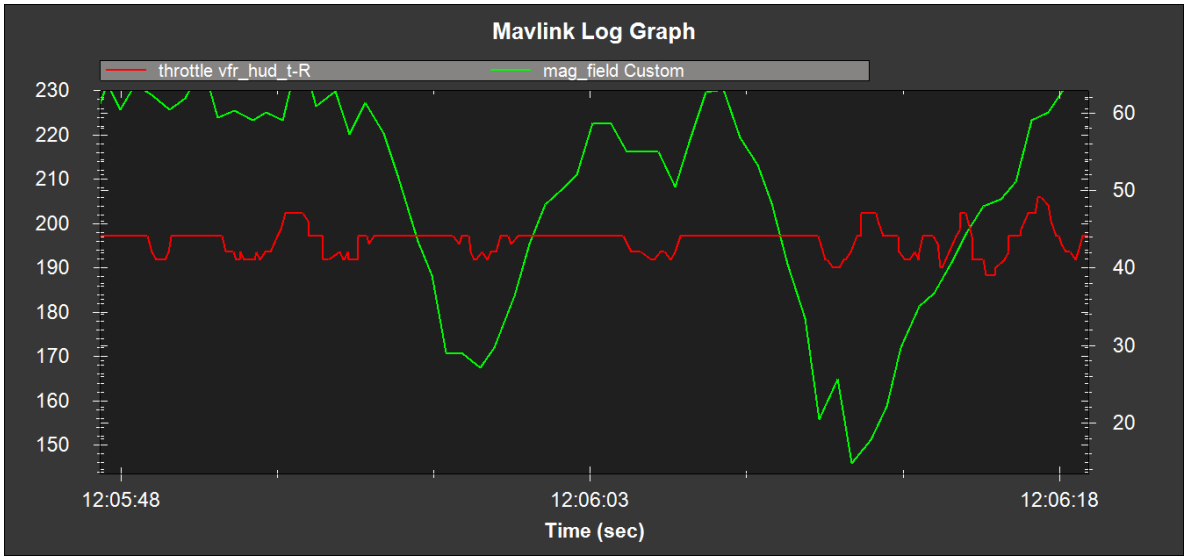


Figura 3.11: Gráfico de campo magnético y aceleración pedida. Se muestra un acercamiento a un descenso inesperado en el campo magnético medido.

Dicha variación representa un 40% de fluctuación. Si bien esto cae dentro del margen, se revisó la corriente medida, lo que se muestra en la Figura 3.12. Se observa que la corriente se mantiene estable en estos descensos, por lo que se descarta ese efecto. Por otro lado, se analiza el efecto de los giros con respecto al eje Z (ángulo de dirección o yaw), mostrándose esa respuesta en la Figura 3.13.

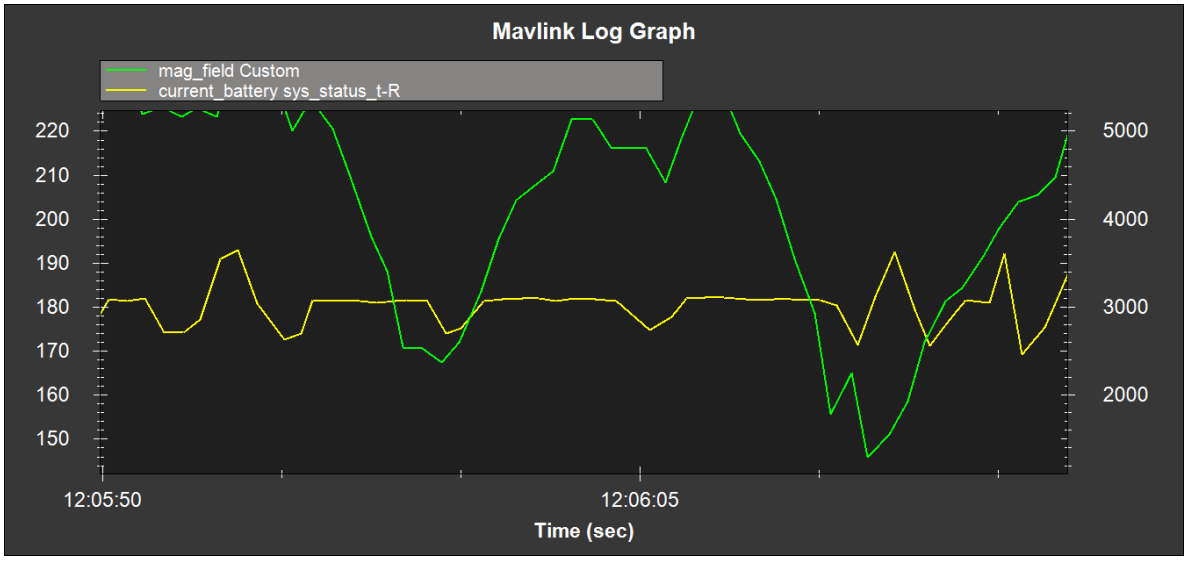


Figura 3.12: Gráfico del campo magnético y corriente circulante. Se observa la misma zona del descenso del campo magnético.

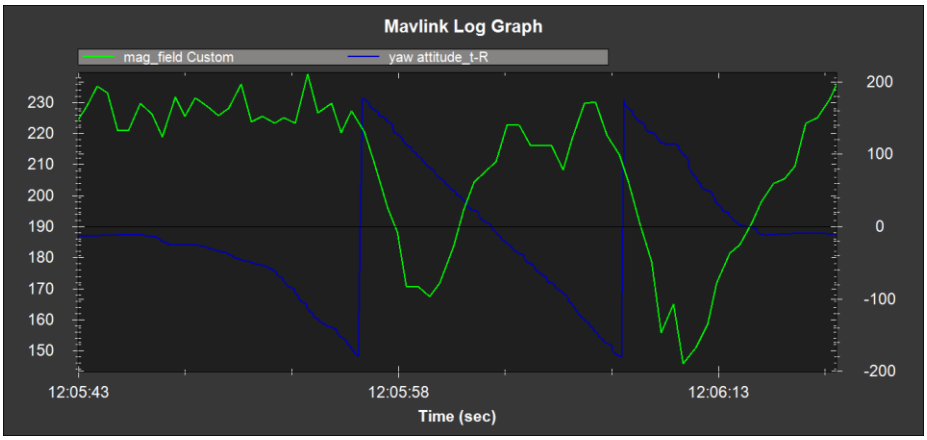


Figura 3.13: Gráfico del campo magnético y el ángulo de dirección. Se observa la misma zona del descenso del campo magnético.

Se observa que los descensos en el campo magnético se producen en la zona en la que las mediciones del ángulo de dirección pasan de valores negativos a valores positivos de manera discontinua, lo que representa giros en 360° que se efectúan durante el vuelo. Esto provoca a su vez las irregularidades en los magnetómetros de los ejes X e Y.

Altitud medida

Para medir la altitud, el *firmware* utiliza tanto la presión barométrica como las mediciones del acelerómetro. Esto se visualiza en la Figura del Anexo XV (pág. 87).

Se observa que en general la altura fluctúa entre 3 y 4 [m], lo que corresponde a la prueba realizada en el laboratorio. Sin embargo, se realizan 2 observaciones:

1. Se puede ver un descenso pronunciado en las mediciones, lo que se muestra en la Figura 3.14. Esto se puede producir debido a diversos factores, sin embargo luego de analizar diferentes variables en los gráficos, se concluye que se debe a una falla en la medición del sensor barométrico.
2. Se observa que la prueba termina en una altitud de ~3 [m]. Habiendo aterrizado la nave al nivel del piso, se concluye que existe un *drift*, es decir, una desviación progresiva en la medición. Esto

también se observa en la Figura 3.15, en donde la nave aterriza 2 veces observándose un ligero aumento en la altitud medida.

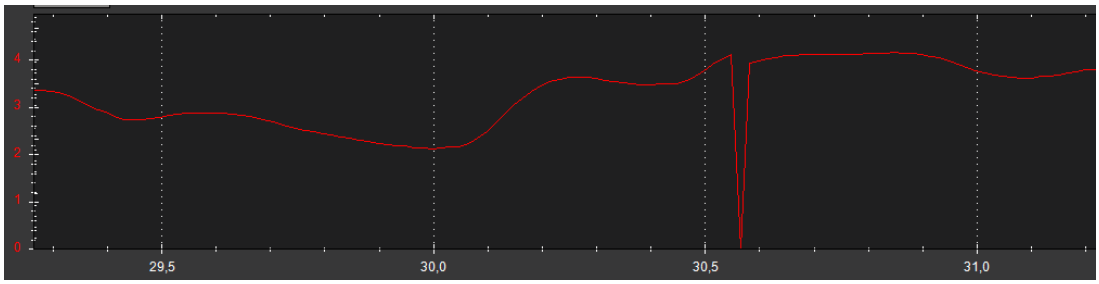


Figura 3.14: Gráfico de la altitud medida, con un acercamiento en la falla ocurrida.

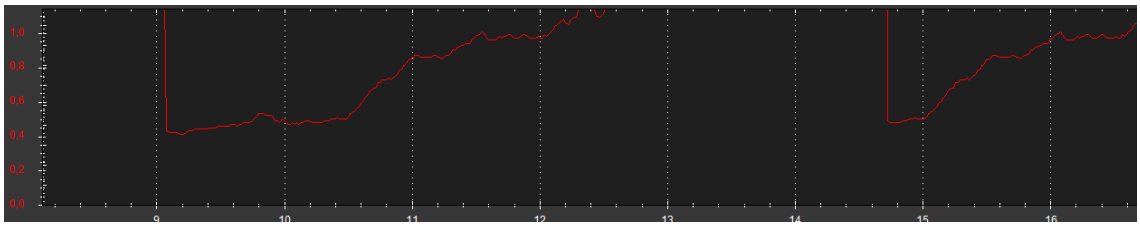


Figura 3.15: Gráfico de la altitud medida, con un acercamiento en la desviación progresiva a la altura del piso.

Se concluye que los problemas encontrados son menores ya que en el caso de pruebas cortas las fallas puntuales se pueden extraer mediante procesamiento de datos. Por otro lado, la desviación progresiva de los sensores es notoria a partir de pruebas superiores a los 2 minutos, lo que no interfiere con el resto de las pruebas.

3.3. Diseño de un asistente de vuelo

3.3.1. La maniobrabilidad de un UAV

El manejo de vehículos aéreos no tripulados presenta numerosas dificultades para los usuarios inexpertos [**31**]. Como se vio en la Sección 2.1, el quad-rotor se trata de un sistema MIMO, en el cual se tienen 6 variables controladas y 4 variables manipuladas, que a nivel bajo corresponden a la velocidad de rotación de cada motor. Los modelos y controladores actuales permiten llevar el manejo de la nave a un nivel de abstracción superior al control directo de los motores, en el cual se controlan 4 variables ligadas a la orientación y la aceleración en el eje vertical de la nave.

La dependencia de las otras dos variables de posición, sumada a los problemas de estabilidad de los UAV, son temas que agregan complejidad al momento de maniobrar. Para poder operar naves con precisión no solo requiere una buena documentación y conocimiento sobre calibración de parámetros, sino que se deben considerar numerosas horas de práctica. Por otro lado, dependiendo del usuario, la experiencia de vuelo puede ser diferente, pudiendo ir desde imposibilidad de elevarse del suelo a niveles de destrucción del equipamiento que impidan su posterior uso. Esto limita tanto la motivación de los usuarios al momento de introducirse al tema como también el desarrollo del uso de vehículos aéreos en la industria al conllevar un costo adicional (tiempo de entrenamiento, salario de un operario experto, repuestos).

Por otro lado, existen modos de vuelo considerados en algunos *firmware* que llevan el control de las naves a un nivel más alto de abstracción, en el que ya no se controla la inclinación directamente sino que se le entregan coordenadas al UAV y éste llega a ese punto de manera autónoma. Para ello se requieren sensores adecuados como localizador GPS u otro tipo de sensor de posicionamiento global o con respecto a referencias. Si bien esta alternativa puede ser atractiva, está limitada al desempeño y disponibilidad de los sensores adicionales (como por ejemplo la señal de un cierto número de satélites), lo que acota su uso a operaciones al aire libre.

El proceso de aprendizaje

En la actualidad, un nuevo operario de UAVs debe superar diversos obstáculos para aprender a volar naves con un nivel de precisión confiable. En primer lugar, se debe tener un conocimiento general del comportamiento de la nave y de los comandos que se le entregan, así como sus consecuencias en su postura. Este conocimiento puede ser teórico o bien puede venir de la experiencia con otras naves o vehículos.

A continuación, se deben realizar pruebas básicas tanto para entender la dinámica y la inercia de la nave, como también para proceder a realizar la sintonización preliminar de parámetros. Este proceso es usualmente lento ya que también contempla la práctica y dominio de los controles. Además, es en esta etapa en donde se producen los accidentes que pueden ralentizar el proceso. Las pruebas constan en primera instancia de instrucciones básicas: ascender verticalmente, mantener una posición estable, desplazarse en alguna dirección a baja velocidad, aterrizar. Eventualmente se deben corregir problemas con la nave misma, como re-calibrar dispositivos, reafirmar componentes al frame, compensar los controles, entre otros.

El sistema de control asistido

Para facilitar el proceso iterativo de aprendizaje, se propone el desarrollo de un **asistente de vuelo**, el cual identifique el grado de experticia del operario mediante un análisis del comportamiento de la nave, y que actúe simplificando su operación en ciertas áreas de especial dificultad. Este sistema tiene como objetivo amortiguar la curva de aprendizaje de este tipo de naves, permitiendo en primer lugar familiarizarse con la dinámica de la nave, separando así las áreas a desarrollar.

3.3.2. ESTRUCTURA GENERAL DEL SISTEMA DE CONTROL ASISTIDO

El sistema de control asistido diseñado contempla tres grandes módulos. El primero consiste en la realización de una prueba de operación básica, en la cual el operario debe seguir una serie de instrucciones de manejo de la nave. A continuación, se realiza un análisis de desempeño de la prueba realizada. Dicho análisis se realiza fuera de línea utilizando los registros de vuelo guardados por el procesador. Utilizando los resultados del análisis, se efectúan finalmente los cambios en el controlador, limitando en mayor o menor grado las referencias admitidas por el sistema. Un diagrama resumen se muestra en la Figura 3.16.

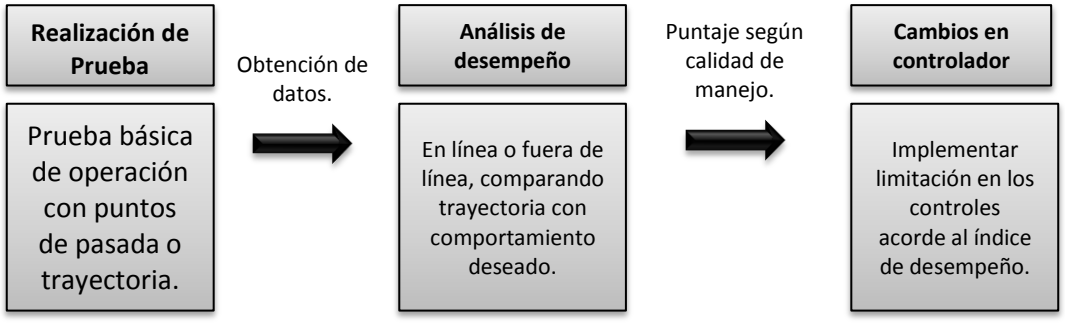


Figura 3.16: Estructura general del sistema de control asistido

Consideraciones previas:

- Mantención de la estructura de los controladores pre-existentes: La separación del controlador en dos niveles tal como se mostró en la Sección 2.4.2 permite simplificar el manejo de la nave, al utilizar referencias correspondientes a ángulos en lugar de sus tasas de cambio, por lo que no se modificaría su estructura general.
- Restricción de referencias del usuario: En pruebas preliminares se observó que cambios bruscos de las referencias angulares en el plano llevan a inestabilidades, pudiendo incluso estrellar la nave

en manos inexpertas. Se consideró actuar a nivel de las referencias permitidas, restringiendo tanto sus valores absolutos así como su tasa de cambio para disminuir inestabilidades.

- Restricciones de los sensores: Para este trabajo, se consideró el manejo de la nave sin restricción en las condiciones de operación, y en particular su uso en el interior del Laboratorio. Esto implicó la indisponibilidad de datos GPS confiables, y la necesidad de métodos alternativos para la determinación de la posición del quad-rotor.

3.3.3. PRUEBA DE OPERACIÓN

Tipo de prueba

La prueba a realizar debe involucrar trayectorias o movimientos deseados puedan mostrar de alguna forma la habilidad del operario al momento de manejar la nave. Se desea poder caracterizar las diferentes competencias del usuario, incluyendo control de aceleración vertical, estabilidad angular, y posicionamiento en el plano X-Y. Por lo tanto, se decidió realizar una prueba en la que el operario debiera ser capaz de despegar y mantener la nave estable en el aire un mínimo de tiempo, para luego proceder a aterrizar.

3.3.4. ANÁLISIS DE DESEMPEÑO

Datos a utilizar

Para decidir qué datos utilizar para el análisis se consideraron los registros, o *'logs'*, que guarda el firmware luego de cada vuelo. Dichos registros incluyen información de diversos tipos como mediciones inerciales, entradas y salidas PWM, niveles de batería, coordenadas GPS, entre otros.

Dadas las condiciones de la prueba, existe una dificultad al momento de determinar las coordenadas en el plano X-Y. Específicamente, al no tener disponibles datos del módulo GPS, se requiere un procesamiento adicional para su obtención. Se exploraron diversas alternativas:

- **Utilización de un sensor externo:** En la actualidad existen alternativas de sensores de posicionamiento como el módulo de **flujo óptico** [**32**]. Estos componentes detectan objetos y marcadores relevantes en el suelo, permitiendo ubicar a la nave con respecto a ellos. Su integración con el sistema existente requeriría una carga de trabajo considerable ya que actualmente no es compatible con el *firmware*, lo que implicaría el diseño de una biblioteca dedicada para un protocolo de comunicación entre el módulo y el controlador.
- **Cálculo de posición por odometría**: La odometría utiliza datos de posición pasados conocidos y la información inercial (orientación + aceleración vertical), en conjunto con un modelo matemático de la nave, para calcular coordenadas estimadas en línea. Debido a la naturaleza ruidosa de las mediciones procedente de las vibraciones de la nave, el cálculo de posición pierde exactitud rápidamente, dejando de ser válido después de pocos segundos.
- **Utilización de un simulador**: El uso de un modelo matemático en conjunto con herramientas computacionales permite una recreación virtual de la prueba, en la cual se pueden obtener de manera estimada todas las coordenadas de la nave. Para poder realizar esto, se debe utilizar modelo en conjunto con las entradas entregadas al controlador en la prueba real. En este caso, el ruido de los sensores no está presente ya que las coordenadas se obtienen de manera directa del modelo y no deben estimarse con un sensor real. Sin embargo, existen imprecisiones debido a las aproximaciones que se realizan en los modelos.

Dada la imprecisión del cálculo por odometría y la extensa carga de trabajo que significa integrar el uso de un sensor externo, además de su alto costo, se decidió utilizar un **simulador** para recrear el vuelo y así obtener datos virtuales sobre la posición de la nave. Para ello, se optó por tomar las órdenes enviadas por el usuario mediante el mando RC, las cuales se encuentran disponibles en los registros, e introducirlas como entradas del simulador. El diagrama de la Figura 3.17 muestra la utilización del simulador en conjunto con la prueba real.

Con esto, se utilizan las estimaciones obtenidas en la prueba real (orientación y altitud) en conjunto con los datos simulados (estimación de la posición en el plano X-Y) para el análisis de desempeño.

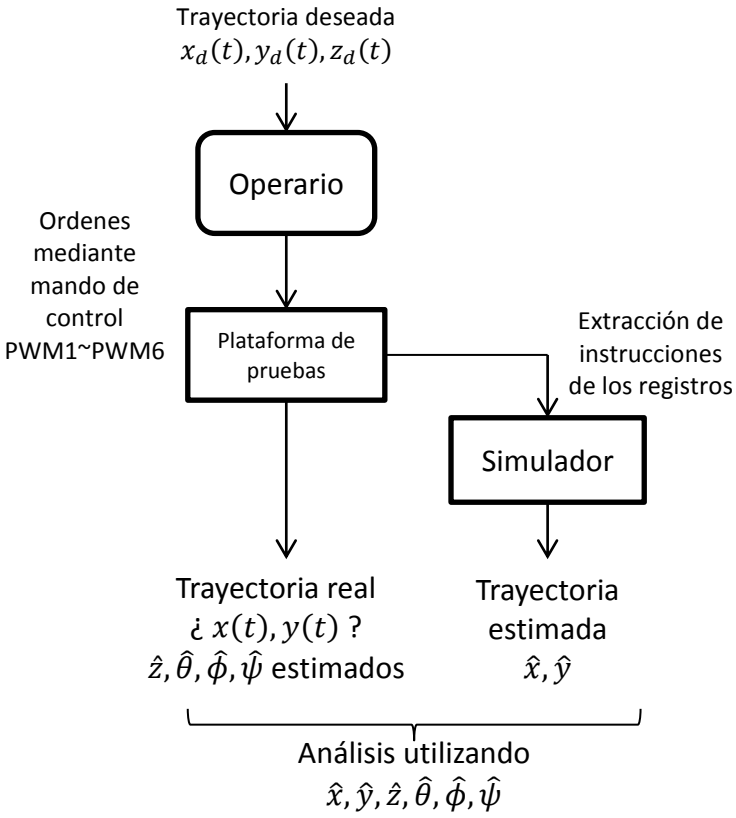


Figura 3.17: Diagrama de bloques que ilustra la obtención de coordenadas de posición estimadas mediante el simulador para un análisis posterior.

Simulador

Se contemplaron tres alternativas para el uso de un simulador:

- El desarrollo de un simulador propio en un lenguaje compatible con los controladores de APM: Copter.
- El uso de un simulador externo adaptando los controladores, con su estructura y parámetros.
- El uso de un controlador existente (SITL), compatible con el código fuente de APM: Copter.

La primera es la que otorga mayor flexibilidad al momento de realizar las simulaciones, sin embargo representa una carga de trabajo considerablemente mayor a las otras. Por otro lado, el uso de simuladores existentes, como modelos de quad-rotors en MATLAB-Simulink o *software* especializados en vuelos simulados, significa migrar muchas de las funciones del *firmware* posiblemente a otros lenguajes.

El simulador SITL [**33**] (*software-in-the-loop*) fue desarrollado por la comunidad de APM como herramienta para análisis del *firmware* APM: Copter y sus demás variantes. Permite la ejecución del código fuente sin necesidad de conectar *hardware*, emulando todos los módulos involucrados (controladores de bajo nivel de los diferentes sensores). Además, funciona realizando una compilación del código completo, el cual ejecuta en conjunto con una simulación de la física de la nave. Dicha simulación se encuentra incluida en el desarrollo de SITL.

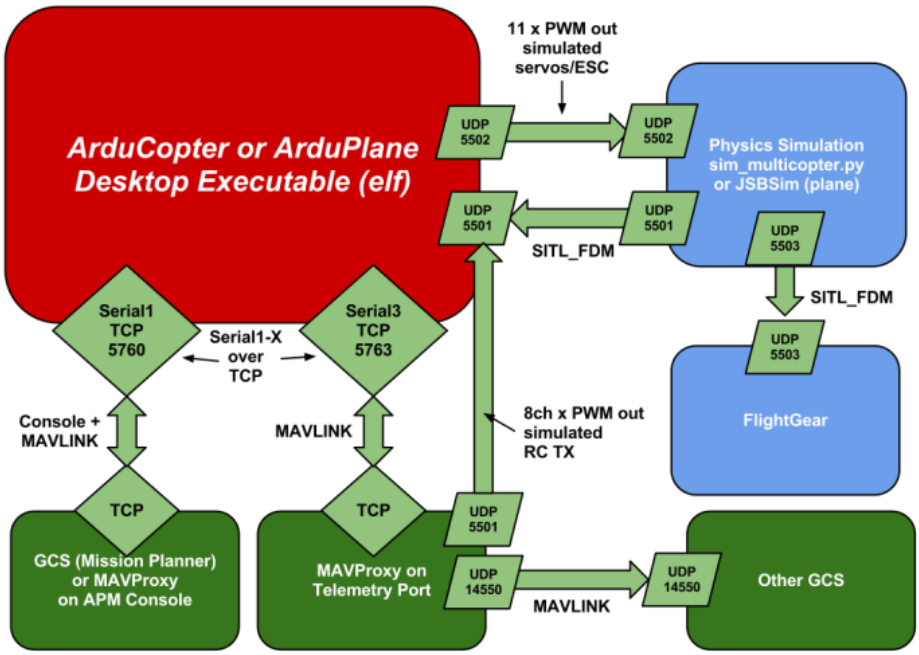


Figura 3.18: Arquitectura del simulador SITL, incluyendo protocolos de comunicación entre diferentes módulos.

Considerando las características explicadas y la carga de trabajo involucrada, se optó por trabajar con el simulador SITL. Para ello, se debió en primer lugar analizar su estructura. La Figura 3.18 muestra los módulos involucrados, incluyendo los puertos utilizados para su comunicación.

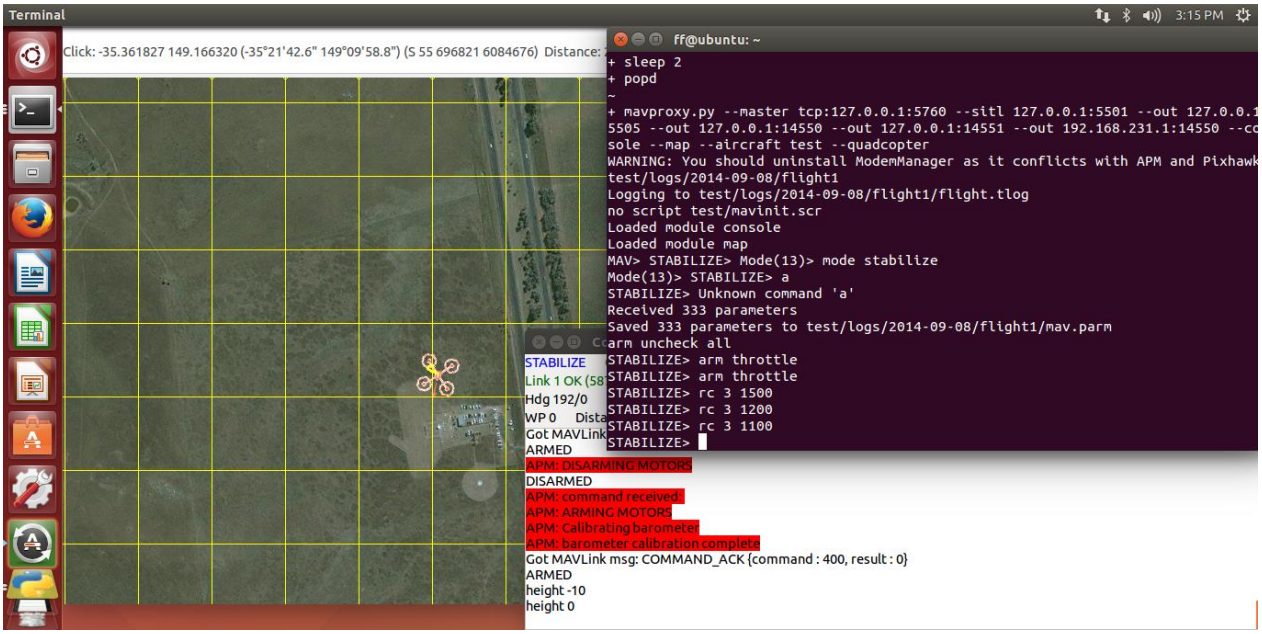


Figura 3.19: Interfaz del programa MAVProxy funcionando con SITL en Ubuntu.

El código fuente se compila en un archivo ejecutable **`ArduCopter.elf`**, el cual se invoca en conjunto con el simulador **`sim_multicopter.py`** que encuentra escrito en lenguaje Python. Estos dos programas interactúan mediante paquetes de datos enviados a través de los puertos UDP 5501-5502. Por otro lado, el programa principal también interactúa, mediante el protocolo de comunicación **Mavlink**, con el programa **MAVProxy**, el cual actúa como interfaz con el usuario, permitiendo el uso de diversas funciones como instrucciones mediante comandos de texto, visualización de un mapa, ejecución de *scripts*, entre otros. La

Figura 3.19 muestra la interfaz MAVProxy, junto con la consola y el mapa virtual, funcionando en conjunto con el simulador SITL.

Para poder simular el vuelo, se utilizó una combinación de funciones existentes en el programa MAVProxy. La primera de ellas es la posibilidad de entregar instrucciones textuales por la consola, pudiendo armar y desarmar los motores, cambiar de modo de vuelo, cargar parámetros, entre otros. Una lista completa se encuentra disponible en la documentación respectiva [**34**]. La segunda fue la integración de bibliotecas ya existentes de Mavlink, las cuales permiten crear instancias de comunicación desde y hacia MAVProxy mediante diversos protocolos como UDP, TCP y conexión serial.

En primer lugar, se tomaron los valores PWM registrados en los archivos **.log** en la prueba realizada, para luego ser procesados y reescritos en el formato de MAVProxy. Dicho formato es de la forma: **"rc x PWM"**, donde **'x'** corresponde al canal de comunicación (entre 1-4 para los 4 canales principales del quad-rotor), y **'PWM'** indica el valor PWM en [ms] de dicho canal. El programa que realiza dicha función se escribió en MATLAB (archivo **rw_rcin.m** en Anexo I).

Una vez ejecutado dicho programa, se obtienen archivos de texto con los comandos a entregar al simulador en cada prueba. Esto se logra utilizando el paquete **pexpect** de Python, el cual ejecuta una aplicación e interactúa con ella de manera automática. Su uso más detallado se puede encontrar en la documentación respectiva [**35**]. Se implementó una línea de comandos basada en un *script* existente desarrollado por los contribuyentes de la comunidad, **sim_arducopter_randy.sh** ubicado en la carpeta ardupilot\Tools\autotest, el cual ejecuta todos los módulos requeridos para correr la simulación SITL.

El *script* desarrollado, **sim_arducopter_ff.sh**, contiene las instrucciones para realizar la simulación completa (ver). En primer lugar, se invoca el programa **rw_rcin.sh** para convertir los datos del archivo de registro. A continuación se compila y se ejecuta el código fuente ubicado en la carpeta ArduCopter, en conjunto con el simulador de la física de la nave, para luego llamar a un segundo programa que ejecuta instrucciones en Python, **mavsim.py.** Este último programa es el encargado de invocar la aplicación MAVProxy.py con todos los módulos asociados e interactuar con ella.

Para ello, se crean 2 instancias: una con el comando **pexpect.spawn('cmd')**, la cual ejecuta el programa indicado en el *string* 'cmd' y se comunica con él, y otra con el comando **mavutil.mavlink_connection('localhost:14551')** para conectarse mediante el puerto UDP indicado al programa principal. Para esta última instancia se utilizó la biblioteca **mavutil**, la cual posee diversas funciones para crear y manipular módulos de comunicación de diversos protocolos, entre otros [**36**].

La aplicación luego procede la inicialización, que consiste en cargar los parámetros del quad-rotor utilizado en la prueba real, armar los motores, y esperar una conexión GPS virtual. Esto se realiza combinando las funcionalidades de pexpect y los mensajes enviados por UDP.

Una vez armados los motores, se procede a comunicar las instrucciones de RC para simular el vuelo. En este punto, se exploraron dos alternativas.

1. *Escritura directa de los valores PWM mediante instrucciones en la consola:*

Se aprovecharon las funciones del paquete pexpect, las cuales permiten enviar comandos textuales al programa invocado mediante la función **send('texto')**. Para ello, se procedió a leer el archivo conteniendo las instrucciones línea por línea, y redirigirlas a la interfaz. Se tiene en consideración que los valores PWM del mando RC son registrados a 50 [Hz], por lo que se utilizó un sistema de espera para reenviarlos a la misma frecuencia. Para asegurarlo, se calcula el tiempo de detención en cada ciclo como sigue:

```
t_1 = t_actual();
…
(Instrucciones)
…
t_final = t_1 + dt;
t_espera = t_final – t_actual();
wait(t_espera);
```

Con esto, el tiempo de procesamiento durante la ejecución del programa se resta al tiempo de espera, lo que permite una simulación más precisa. La implementación de esta solución se encuentra en el Anexo III bajo el nombre **`mavsim_pexpect.py`**.

2. Envío de valores PWM mediante el puerto UDP

Se exploró la documentación del protocolo MAVLink [**37**], en donde existen diversas funciones para comunicarse mediante puertos UDP. Más precisamente, al crear una instancia con la función `mavlink_connection`, ésta permite enviar información directamente al programa al que está enlazada, evitando la necesidad del empaquetamiento de datos. Esto último se puede realizar utilizando la función **`rc_channels_override_send`**. Esto se aprovechó implementando la función auxiliar `set_attitude(rc1, rc2, rc3, rc4)`, la cual envía al programa principal los valores entregados como argumentos para ser asignados como entradas RC de los 4 canales principales.

El tiempo de espera se implementó de la misma manera que en el caso anterior. La solución se encuentra en el Anexo IV bajo el nombre **`mavsim_udp.py`**.

Con cualquiera de las dos soluciones, al terminar la simulación los datos del vuelo virtual quedan guardados en formato `.tlog` en una carpeta con la fecha de la ejecución del programa.

Procesamiento de datos

El análisis de desempeño se debe realizar analizando los datos de vuelo con métricas de calidad de manejo, las cuales se determinan a partir de la prueba de operación. Se debe poder determinar con qué precisión se logró seguir las instrucciones que se dieron.

Para trabajar con los valores de los registros, en primer lugar se deben exportar a un formato en donde sean de más fácil acceso para ser manipulados. Se escogió trabajar con MATLAB ya que Mission Planner tiene la opción de obtener los datos en formato **`.mat`**. Esto se logra en el menú **Telemetry logs** de la pestaña Flight Data, luego seleccionando **Tlog > Kml or Graph** y finalmente **Create MATLAB file**.

Se escribió el *script* **`analyze_logs.m`** para procesar dichos datos (ver Anexo V). En él, en primer lugar se cargan los datos reales de los registros de telemetría y se extraen los vectores correspondientes a la postura de la nave (3 ángulos de orientación), además de la altitud barométrica.

Por otro lado, se cargan los datos simulados de los registros de telemetría obtenidos a través de la simulación realizada con SITL y MAVProxy, en donde se extraen los datos de posicionamiento global GPS. Dichos registros son del tipo **`int`** representando grados decimales con 7 cifras de precisión, las cuales deben ser multiplicadas por 10^{-7} para obtener su valor decimal real.

Para convertir estos datos en unidades de uso práctico, se recurre a la definición del largo de un grado de latitud y de longitud siguiendo el modelo WGS84 de la tierra, el cual es utilizado por los sistemas GPS actuales, y las proyecciones Mercator [**38**]. Con esto, se puede aproximar el largo de un grado (1°) de latitud y de longitud, en metros, con las siguientes ecuaciones:

$$l_{lat} = \frac{\pi a(1-e^2)}{180\left(1-e^2(\sin^2\phi)\right)^{\frac{3}{2}}}$$

$$l_{lon} = \frac{\pi \cos\phi}{180\left(1 - e^2(\sin^2\phi)\right)^{\frac{1}{2}}}$$

donde e representa la excentricidad de la tierra modelada como un elipsoide, y ϕ corresponde a la latitud en donde se quiere calcular dichas distancias.

Se implementaron estas funciones en MATLAB, bajo los nombres **`length_latdeg.m`** y **`length_londeg.m`** (ver Anexos VI y VII). Utilizando los largos de 1° de latitud y longitud, se pueden obtener aproximaciones, en metros, de los desplazamientos de la nave en los ejes X-Y de la tierra multiplicando esos valores base por las coordenadas obtenidas:

$$x_t(m) = x_t(\text{deg}) \cdot l_{lat}\left(\frac{m}{\text{deg}}\right)$$

Análogamente,

$$y_t(m) = y_t(\text{deg}) \cdot l_{lon}\left(\frac{m}{\text{deg}}\right)$$

Una vez convertidas las unidades, se posiciona el origen del sistema de referencia en el valor inicial de los vectores para visualizar los desplazamientos de manera más directa. Esto se realiza restando el valor inicial de cada señal a la señal completa.

Se calcularon características básicas de todas las señales, como media, desviación estándar, máximo y mínimo. Además, se escribieron dos funciones auxiliares para ayudar con el análisis de la prueba:

- **`count_crossing.m:`** Recibe como argumentos un vector conteniendo una señal, y un umbral. Entrega como resultado el número de veces que la señal transgredió el umbral, alejándose desde el origen.
- **`time_over.m`**: Recibe como argumentos un vector conteniendo una señal, y un umbral. Entrega como resultado los índices entre los cuales la señal superó el umbral indicado. Si existe más de una transgresión, se considera la más extensa.
- **`split_logs.m`**: Toma los sets de datos de diferentes pruebas realizadas, los cuales se extraen a partir de los registros de vuelo (.log) y transformados con el *software* Mission Planner, y los guarda con un formato específico para poder ser procesados posteriormente por el programa de análisis de las pruebas.

Dichas funciones se encuentran en los Anexos VIII y IX respectivamente.

Criterios de calificación

Para calificar el desempeño del operario, el análisis se divide en tres categorías diferentes: Estabilidad en el plano X-Y, dictada por el control de los ángulos *pitch* y *roll*; estabilidad de altitud, lo que se relaciona con el control de aceleración; y por último la precisión en la ubicación en el plano X-Y. Para cada uno de estos criterios, se consideraron diferentes factores que influyen en el puntaje parcial, los que se detallan a continuación.

Pitch-Roll:

La estabilidad de los ángulos de elevación y alabeo constituyen la base de la estabilidad aérea. Para evaluar el manejo de la nave, se consideran varios aspectos. En primer lugar, dada la naturaleza de la prueba, se espera que los ángulos se mantengan lo más cercanos a 0 que sea posible. Para ello, se cuenta el número de transgresiones de ambos ángulos mayores a 5° con la función **`count_crossing.m`** mediante el siguiente código:

```
for i = 1:10
    pitch_count(i) = count_crossing(abs(pitch(i)),i);
    roll_count(i) = count_crossing(abs(roll(i)),i);
```

Se consideró el umbral de 5° como máximo a partir de datos de vuelos anteriores. Para determinar un puntaje, se siguió la siguiente regla para cada ángulo:

- $cruces_{5°\,pitch/roll} \leq 5 \Longrightarrow score_{pitch/roll} = 5$
- $5 < cruces_{5°pitch/roll} < 10 \Longrightarrow score_{pitch/roll} = 10 - cruces_{5°\,pitch/roll}$
- $cruces_{5°pitch/roll} \geq 10 \Longrightarrow score_{pitch/roll} = 0$

Con esto, el puntaje de esta categoría queda dado por la suma de los puntajes parciales:

$$score_{pr} = score_{pitch} + score_{roll}$$

Por otro lado, se consideró que una desviación estándar elevada de las señales también significa una mala estabilidad, por lo que se penalizó cada puntaje de la siguiente manera:

- $\sigma_{pitch/roll} > 2° \Longrightarrow score_{pitch/roll} = \frac{score_{pitch/roll}}{2}$

Aceleración (throttle)

La estabilidad de la aceleración vertical es igual de importante que la estabilidad angular, ya que el operario debe ser capaz de mantener la nave a una altitud relativamente estable, así como despegar y aterrizar con experticia. Para determinar el desempeño en esta área, el puntaje se dividió en dos: tasa de ascenso y descenso (o *climb-rate*), y altitud máxima.

1. Para el *climb-rate*, se consideraron en primer lugar umbrales bajo los cuales se debe operar al manejar la nave de manera segura. Los valores límites se determinaron a partir de datos de vuelos anteriores. Se siguió la siguiente regla:
   - $climb_{max} = \max(abs(climb\_rate))$
   - $climb_{max} > 2\frac{m}{s} \Longrightarrow score\_climb = 0$
   - $climb_{max} > 1\frac{m}{s} \Longrightarrow score\_climb = 2.5$
   - $climb_{max} < 1\frac{m}{s} \Longrightarrow score\_climb = 5$

Por otro lado, se consideró el número de intentos de despegue, lo que generalmente muestra la familiaridad del operario con la dinámica de la aceleración de la nave. Se determinó que cada intento de despegue corresponde a una medición en la altitud que supera el umbral de 0.5 [m]. Se penalizó con 1 punto por cada intento más allá de 1 intento, llegando a un mínimo de 0 puntos.

2. Para la altitud máxima, se consideran las instrucciones de la prueba en las cuales se especifica que se mantenga una altura de aproximadamente 3 [m], agregándose un margen de ±1 [m] debido a posibles imprecisiones del sensor y perturbaciones. La regla fue la siguiente:
   - $alt_{max} < 1 \vee alt_{max} > 5 \Longrightarrow score\_alt = 0$
   - $alt_{max} < 2 \vee alt_{max} > 4 \Longrightarrow score\_alt = 2.5$
   - $2 < alt_{max} < 4 \Longrightarrow score\_alt = 5$

Con esto, el puntaje de esta categoría queda dado por la suma de los puntajes parciales:

$$score\_throttle = score\_climb + score\_alt$$

Posicionamiento

Para esta sección se utilizaron los datos simulados de GPS. Al igual que para las otras categorías, se basó en dos criterios. El primero es no alejarse demasiado del punto de inicio, lo que se traduce en una distancia máxima al origen menor a cierto umbral. El segundo es que el punto de aterrizaje sea cercano al punto de despegue.

Se encontró el punto más alejado del origen en latitud y longitud, en metros, y se calculó su distancia al origen:

$$r_{max} = \sqrt{lat_{max}^2 + lon_m^2}$$

El criterio para asignar puntaje fue el siguiente:

- $r_{max} < 5 \Longrightarrow score\_rmax = 5$
- $r_{max} > 5 \Longrightarrow score\_rmax = 5 \cdot f(r_{max})$,

donde la función de decaimiento $f(x)$ viene dada por:

$$f(x) = \exp\left(-\frac{1}{2}(x-5)\right)$$

La Figura 3.20 muestra la función de decaimiento. Se escogió usar una función de decaimiento y no un cambio discreto debido a la imprecisión de la simulación. El puntaje decae a 0.5 a los 10 metros de distancia máxima obtenida.

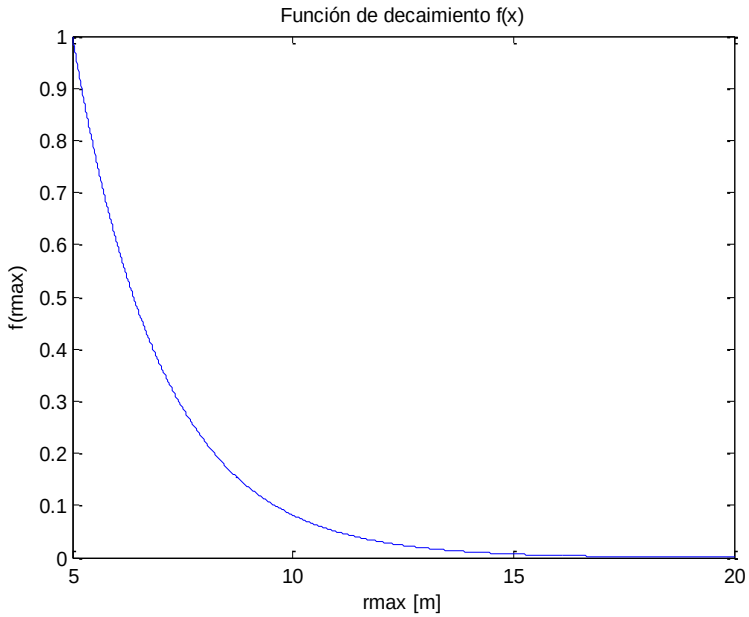


Figura 3.20: Función de decaimiento $f(x)$ utilizada para el puntaje del posicionamiento máximo obtenido por la simulación.

Por otro lado, para el punto de aterrizaje se siguió la misma idea, calculándose la distancia final al origen como:

$$r_{final} = \sqrt{lat_{final}^2 + lon_{final}^2}$$

Con esto, la regla para puntuar fue la siguiente:

- $r_{final} < 1 \Longrightarrow score\_rfinal = 5$
- $r_{final} > 1 \Longrightarrow score\_rfinal = 5 \cdot g(r_{final})$

donde nuevamente la función $g(x)$ es una función de decaimiento, esta vez definida como:

$$g(x) = \exp(-(x-1))$$

La Figura 3.21 muestra la función de decaimiento. El puntaje decae a 0.5 a 3.3 [m] del origen.

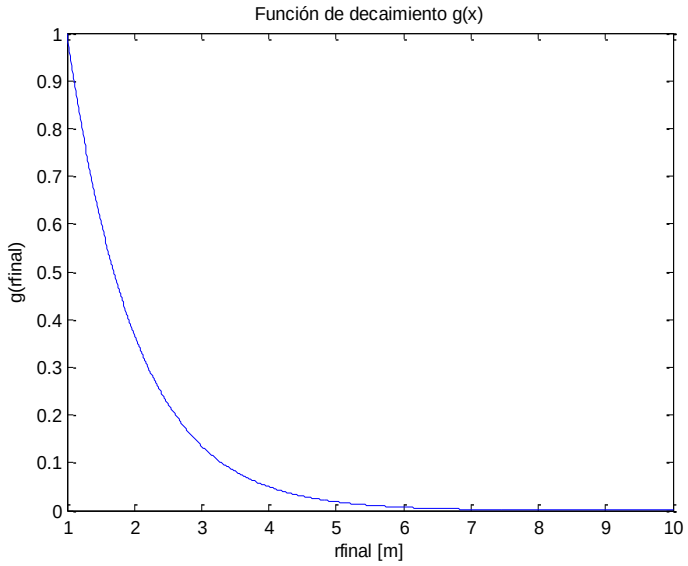


Figura 3.21: Función de decaimiento $f(x)$ utilizada para el puntaje del posicionamiento final obtenido por la simulación.

Finalmente, la puntuación total de la categoría viene dada por:

$$score\_pos = score\_rmax + score\_rfinal$$

Duración de la prueba

Si bien se usaron criterios específicos para cada puntaje, se utilizó la función **`time_over`** para medir el tiempo de la prueba pasado sobre 1 [m] y 2 [m], ya que una prueba demasiado corta podría otorgar falsos altos puntajes. Se utilizó el siguiente criterio:

- $time\_over(1[m]) < 15\ [s] \Longrightarrow score = \frac{score}{2}$
- $time\_over(2[m]) < 10\ [s] \Longrightarrow score = \frac{3}{4} score$

Puntajes finales

Una vez obtenidos los puntajes de las 3 categorías, se debe efectuar una suma ponderada de ellos. Los pesos deben ser calibrados en función de los resultados dependiendo de los siguientes factores:

- Deben reflejar la calidad de manejo del usuario.
- Se le debe dar más peso a los resultados más confiables (i.e. de mayor precisión).

3.3.5. MODIFICACIÓN DEL CONTROLADOR

Características generales de APM: Copter

El código fuente se encuentra escrito en lenguaje C++, y consta de diversas bibliotecas, las cuales cumplen diferentes funciones incluyendo control de la nave (funciones relativas a controladores), manejo de periféricos, navegación, conversión de datos entrada/salida, entre otros. Dichas bibliotecas son compartidas por todas las versiones de APM (Copter, Plane y Rover, para multicópteros, aeroplanos y vehículos terrestres, respectivamente).

Por otro lado, una porción del código de APM: Copter es exclusivo para su uso en multicópteros, e incluye los programas principales y las configuraciones de modos de vuelo, así como la declaración de variables. Todo el código se encuentra disponible para su uso y modificación bajo los términos de servicio de GitHub [**39**].

La implementación de un sistema de control asistido se realizó mediante un procesamiento fuera de línea de datos. Sin embargo, como se mencionó en 3.3.2, el asistente contempla una modificación de los controladores a nivel de los límites de las referencias. Para ello, se observó la arquitectura del control en APM: Copter.

En primer lugar, el programa principal se encuentra escrito en el archivo **`ArduCopter.pde`**, en donde se declaran numerosas variables y se llama al *loop* principal de funcionamiento. Dicho módulo corre a 100 [Hz], en donde se ejecutan las siguientes tareas:

1. Lectura de mediciones inerciales y transformación de coordenadas.
2. Lectura de módulos externos como cámaras.
3. Ejecución de controladores de bajo nivel, requiriendo solo valores pasados de referencias y lecturas de mediciones inerciales.
4. Escritura de salidas PWM a los motores.
5. Lectura de entradas PWM del mando RC
6. Ejecución de controladores de alto nivel dependiendo del modo de vuelo.
7. Actualización de referencias para controladores de bajo nivel.

La tarea que concierne las limitaciones de las referencias corresponde a la ejecución de controladores de alto nivel, los cuales se dividen en controladores del ángulo de dirección (*yaw*, ψ), de los ángulos de alabeo y elevación (*roll-pitch*, $\theta - \phi$), y de la aceleración vertical (*throttle*, u).

Modificación del firmware

En la estructura del *firmware* se contempla la implementación de diversos **modos de vuelo**, los cuales utilizan a su vez **modos de control** para cada una de las variables involucradas (*roll-pitch*, *yaw*, aceleración, y navegación automática). Para ayudar a la estabilización en el plano X-Y, se sigue una serie de pasos:

1. Copia del modo de vuelo estabilizador en un modo de vuelo no utilizado para poder modificarlo.
2. Copia del modo de control para los ángulos *roll–pitch* en el nuevo modo de vuelo.
3. Modificación del nuevo modo de control *roll-pitch* para reducir los ángulos máximos y la tasa de cambio de los ángulos de referencia en función de parámetros modificables.

Se analizó la estructura de los modos de vuelo en el código fuente de APM: Copter. Se escogió el modo de vuelo SPORT para ser modificado. En primer lugar, se asoció el modo SPORT a todos los modos de control utilizados por el modo STAB, además de copiar la inicialización. Esto se realizó modificando los archivos **`ArduCopter.pde, system.pde`** y **`APM_config.h`**.

En segundo lugar, se creó un nuevo modo de control *roll-pitch* en base al original. Para ello, se copió el modo original en el modo ROLL_PITCH_SPORT en el archivo **`ArduCopter.pde`**, asociando nuevamente el modo SPORT al recién creado en **`APM_config.h`**.

A continuación, se modificó el nuevo modo de control ROLL_PITCH_SPORT para incluir saturación de ángulos de referencia. Esto se realizó interviniendo las funciones del archivo **`Attiude.pde`**, las cuales son funciones de cálculo llamadas por los controladores en cada ciclo.

El diagrama de la Figura 3.22 muestra la secuencia de funciones que se ejecutan en los controladores de alto nivel con el fin de obtener referencias para los controladores de bajo nivel.

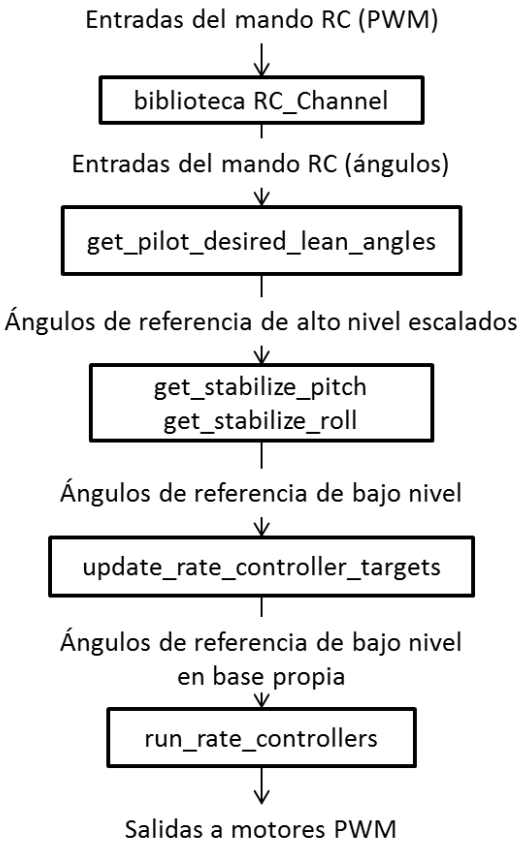


Figura 3.22: Diagrama ilustrativo mostrando las funciones encargadas de procesar desde las entradas PWM del control RC hasta las salidas PWM hacia los motores.

Como se puede ver, en primer lugar la función **`get_pilot_desired_lean_angles`**, ubicada en el archivo **`Attitude.pde`**, transforma entradas del mando RC en ángulos de referencia. Para ello, en la función se realiza un escalamiento en función de un parámetro. Una vez obtenidas las referencias de ángulos, las funciones **`get_stabilize_pitch/roll`** realizan el cálculo de los controladores de alto nivel, obteniendo como resultado referencias para los controladores de bajo nivel. Esto se calcula utilizando las ganancias K_P de los ángulos considerados. Una vez ejecutada dicha función, se pasan las referencias como entrada a los controladores de tasa de cambio (2.9).

Para limitar las referencias y su tasa de cambio tal como se indicó en 3.3.2, se definieron dos parámetros:

- **`sat_angles:`** límite de saturación para las referencias del controlador de alto nivel.
- **`sat_angles_deriv:`** límite de saturación de la tasa de cambio para las referencias del controlador de alto nivel.

Estos parámetros fueron declarados en los archivos **Parameters.pde** y **Parameters.h**, junto con sus valores por defecto en el archivo **config.h**. Además, en el archivo **ArduCopter.pde** se declararon las variables auxiliares **prev_target_roll/pitch** para guardar los valores anteriores de las referencias.

Estos valores se utilizaron para redefinir la función **get_pilot_desired_lean_angles:**

- **get_pilot_desired_lean_angles:** Se elimina el escalamiento de la referencia dependiente de un parámetro. Dicha funcionalidad que, si bien limita los movimientos, es inconveniente ya que altera la sensibilidad de los controles. Por otro lado, tratándose de un parámetro global para el modo STAB, se prefiere independizar su uso al del asistente de vuelo. Por lo tanto, se optó por entregar directamente los ángulos al controlador. Se agregó una limitación de dos partes a la entrada en función de los parámetros `sat_angles` y `sat_angles_deriv`. Un pseudo-código se presenta a continuación:

```
// Saturación lineal
angulo_referencia = limitar_entre(angulo_entrada,-sat_angles,+sat_angles);
// Saturación de tasa de cambio
angulo_referencia = limitar_entre(angulo_referencia,angulo_previo – sat_angles_deriv,
angulo_previo + sat_angles_deriv)
// Guardar valor para siguiente iteración
angulo_previo = angulo_referencia;
```

En la Tabla 3.4 se resumen los cambios hechos en los diferentes archivos. Debido a su extensión, los códigos no se encuentran anexados. Sin embargo, están disponibles en una rama del proyecto en el repositorio Github para su acceso [**40**].

Tabla 3.4: Resumen de cambios realizados en el código fuente del firmware APM: Copter en la implementación del sistema de control asistido.

Nombre archivo	Funciones importantes	Cambios realizados
`Arducopter.pde`	Inicializaciones y programa principal, instrucciones de modos de control, declaración de variables auxiliares, asociación de modos de control a funciones de los controladores.	Declaración de variables auxiliares utilizadas, asociación de modo de control SPORT *roll-pitch* a nuevas funciones del controlador.
`Attitude.pde`	Cálculos de los controladores, funciones utilizadas por los modos de control, funciones auxiliares de transformación de unidades, controladores de bajo nivel.	Nueva función get_pilot… asociada al modo de control SPORT *roll-pitch*.
`config.h`	Valores numéricos por defecto de ganancias y parámetros.	Definición de los valores por defecto de los dos parámetros agregados.
`Parameters.pde`	Listado de parámetros para su despliegue en el listado completo en Mission Planner.	Se agregaron los dos nuevos parámetros para su modificación en el *software*.
`Parameters.h`	Declaración de parámetros.	Se declararon los nuevos parámetros.
`APM_config.h`	Definiciones de adicionales opciones adicionales.	Se asociaron los modos de control del modo SPORT a los del modo STAB, excepto por *roll-pitch*.

Los valores numéricos de los dos parámetros creados debieron ser calibrados en función de las pruebas realizadas. Sus unidades son las siguientes:

- **`sat_angles:`** centi-grados (°/100).
- **`sat_angle_deriv:`** centi-grados por ciclo, donde la porción de código considerada se ejecuta a 100 [Hz], por lo que el parámetro representa directamente la tasa de cambio máxima permitida de la referencia en grados por segundo (°/[s]).

Capítulo 4: Resultados

En este capítulo se presentan las diferentes pruebas realizadas en conjunto con los resultados obtenidos en forma de gráficos y tablas. Las Figuras referenciadas en Anexo se encuentran al final del texto. En primer lugar, la Sección 4.1 presenta las pruebas del código modificado para ilustrar su correcto funcionamiento acorde a los requerimientos. A continuación, en la Sección 4.2 se muestran los resultados de pruebas de operación realizadas por diferentes usuarios, así como las simulaciones realizadas en base a los datos obtenidos. Se indican además los índices de desempeño obtenidos utilizando los criterios definidos, así como la definición de los valores de los parámetros del asistente. Finalmente, en la Sección 4.3 se presentan las nuevas pruebas de vuelo realizadas con el sistema de control asistido en funcionamiento y sintonizado a partir de los resultados de las pruebas de operación para los mismos usuarios, incluyendo un análisis comparativo.

4.1. Prueba del código modificado (modo SPORT)

Para verificar el funcionamiento del *firmware* modificado (ver la Sección 3.3.5), se probaron las referencias calculadas por los controladores a partir de los comandos entregados por el operario. Esto se logró mediante pruebas sin elevar la nave. Las Figuras se encuentran en el Anexo XVI (pág. 88).

Se observa que las referencias no superan el límite establecido (5° en la primera prueba). Por otro lado, se calculó para la segunda prueba la tasa de cambio de la referencia a obtener, lo que entrega un máximo de $10° / s$ para cada ángulo. Las Figuras siguientes muestran los cambios en las referencias en conjunto con los cambios ordenados, en unidades PWM de entrada para cada canal correspondiente. Para analizar los valores de manera más exacta, se graficaron las mismas variables en MATLAB (Figura 4.1). Se observó, utilizando la herramienta *data cursor*, que la tasa de cambio fue de 10° en exactamente 1 [s], lo que corresponde a lo esperado.

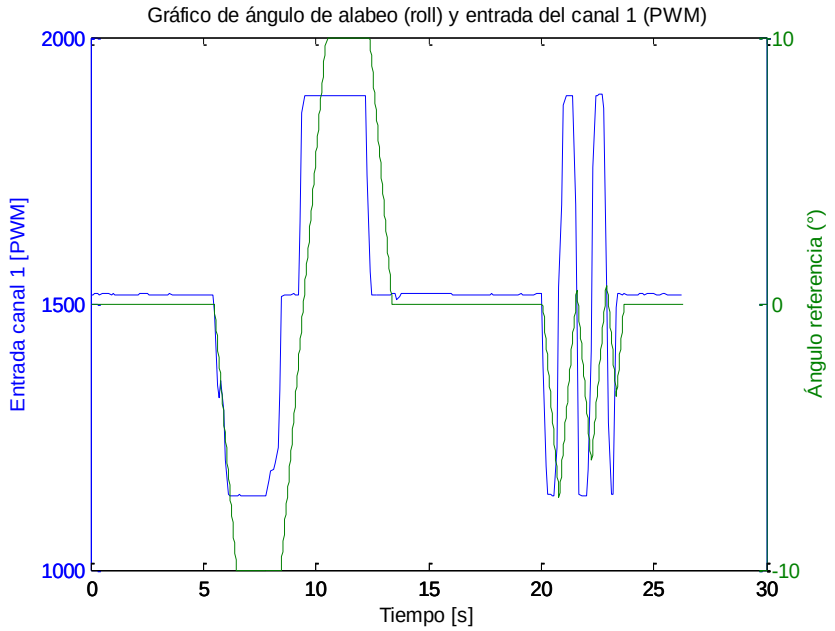


Figura 4.1: Gráfico del ángulo *roll* junto con la entrada del canal 1 en PWM.

4.2. Pruebas de operación y evaluación de desempeño

Se realizaron pruebas de operación con 4 operarios, los cuales presentaban diferentes niveles de experiencia de manejo de UAVs. Además, cada uno realizó la prueba más de 3 veces para poder obtener resultados de mayor confiabilidad y descartar efectos externos, obteniéndose un resultado final

promediado a partir de todas las pruebas. Todas las pruebas se efectuaron en el interior del laboratorio, bajo las mismas circunstancias y condiciones ambientales. En el caso de obtenerse pruebas fallidas, cuya determinación se realizó en base a una altitud máxima menor a 0,5 [m], se descartaron sus resultados.

4.2.1. OPERARIOS

Se consideraron 4 operarios diferentes, cuyos detalles se indican en la Tabla 4.1. Los niveles de experiencia previa deberían reflejarse en los resultados obtenidos, y más específicamente en los puntajes de las evaluaciones.

Tabla 4.1: Operarios que realizaron las pruebas de operación.

Operario	Nombre	Nivel de experiencia
1	Rodrigo	Alta
2	Paul	Baja
3	Isao	Media
4	Felipe	Media

4.2.2. RESULTADOS DE LAS PRUEBAS

En el Anexo XVII (pág. 90) se encuentran los gráficos de las pruebas para los 4 operarios. Existen diferencias en los desempeños, lo que es esperable debido a la diferencia entre su experticia previa a la realización de las pruebas. Las Tablas 4.2 a 4.5 muestran los puntajes para las dos primeras categorías de cada prueba.

Tabla 4.2: Puntajes obtenidos para el primer operario en las dos primeras categorías.

Prueba	***Pitch-Roll***	**Aceleración**
1	5	10
2	10	10
3	10	10
4	10	7,5
5	10	10
6	10	7,5
Promedio	**9,2**	**9,2**

Tabla 4.3: Puntajes obtenidos para el segundo operario en las dos primeras categorías.

Prueba	***Pitch-Roll***	**Aceleración**
1	5	3
2	5	1,5
3	5	2
4	4,5	3,75
Promedio:	**4,9**	**2,6**

Tabla 4.4: Puntajes obtenidos para el tercer operario en las dos primeras categorías.

Prueba	***Pitch-Roll***	**Aceleración**
1	2,5	2,5
2	4,125	6,75
3	2,5	2,5
4	7,5	4,875
5	4,5	3,75
Promedio:	**4,2**	**4,1**

Tabla 4.5: Puntajes obtenidos para el cuarto operario en las dos primeras categorías.

Prueba	*Pitch-Roll*	Aceleración
1	5	5
2	5	2
3	4	2,75
4	6,75	7,5
5	0	2,5
Promedio:	**4,2**	**4,0**

Se destaca que para el tercer operario, en las pruebas 1 y 3 se tuvieron aterrizajes muy agresivos, lo que se refleja en los puntajes obtenidos.

4.2.3. SIMULACIONES Y ANÁLISIS

Como se indicó en la Sección 3.3.4, se programaron 2 métodos diferentes para el envío de comandos al programa encargado de realizar las simulaciones. Se probaron inicialmente ambas alternativas.

Envío de commandos de texto mediante el paquete *pexpect*

La primera alternativa de simulación consistió en el uso del archivo **`mavsim_pexpect.py`**, utilizando mensajes textuales enviados automáticamente a la interfaz MAVProxy. Este método, presento diversos problemas, siendo el más importante la detención del envío de los comandos, visible directamente en la consola, al cabo de 30 [s] de simulación. Esto mostró una poca confiabilidad en este método, por lo que finalmente se descartó.

Envío de commandos mediante un puerto UDP

El uso del archivo **`mavsim_udp.py`** logró generar registros de un vuelo simulado sin mayores problemas. El *script* fue ejecutado hasta el final, guardando los archivos de registros **`.tlog`** para luego ser procesados por el archivo de análisis en MATLAB.

Análisis de resultados

En el Anexo XVIII se muestran las respuestas obtenidas a partir de las simulaciones realizadas para todas las pruebas de los distintos operarios. Se realizó la conversión de grados a metros indicada en la Sección 3.3.4 (pág. 35). Además, se comprobó que la duración de las pruebas simuladas fue la misma que para las pruebas reales, mostrándose en los gráficos la escala de tiempo utilizada.

En primer lugar, destaca la magnitud de los resultados, los cuales llegan a valores de hasta ±100 [m]. Dado que la prueba se realizó en el interior de un laboratorio, es directa la imprecisión de estos resultados en comparación con los datos reales. Se observa además que la diferencia crece considerablemente con la duración de la prueba, lo que se aprecia especialmente en los primeros dos operarios. En el primer caso, las pruebas son de una duración de ~70 [s], mientras que el segundo operó la nave durante ~40 [s], en promedio.

Por otro lado, también se destaca la diferencia de altitudes alcanzadas en las diferentes pruebas, lo que también puede considerarse un factor influyente. De existir demasiadas diferencias entre el modelo de la nave presente en el simulador y el vehículo real, específicamente en su peso y potencia de motores, la diferencia proporcional aumenta al exigir la nave en mayor medida.

Evaluando las simulaciones utilizando los criterios indicados en la Sección 3.3.4, se obtuvieron puntajes para cada prueba. Los valores calculados fueron inferiores a 1 para todas las pruebas, lo que refleja la poca precisión del simulador. Debido a estos resultados, se consideró que su uso en el cálculo de parámetros del

asistente de vuelo sería perjudicial ya que no representan los desplazamientos de la nave de manera realista.

Calibración de parámetros

Una vez obtenidos los puntajes de cada prueba, se decidió realizar una suma ponderada de ellos para obtener un puntaje global para cada usuario como se mencionó en la Sección 3.3.4. Esto se diseñó tomando en cuenta diversos factores, siendo el más importante la imprecisión mostrada por los datos de posicionamiento GPS simulados. Se utilizó como criterio principal que el usuario experto no necesitara el uso del asistente de vuelo. Por lo tanto, se particionó el espacio de la siguiente manera:

- Puntaje ponderado mayor a 8: El usuario no necesita el sistema de control asistido.
- Puntaje ponderado menor a 8: Los parámetros se deben calibrar siguiendo las siguientes fórmulas:

$$sat\_angles = \begin{cases} 1000 & \text{si } 6 \leq score \\ (score - 2) \cdot \dfrac{700}{4} + 300 & \text{si } 2 \leq score < 6 \\ 300 & \text{si } score < 2 \end{cases}$$

$$sat\_angle\_deriv = \begin{cases} 10 & \text{si } 6 \leq score \\ (score - 2) \cdot \dfrac{5}{4} + 5 & \text{si } 2 \leq score < 6 \\ 5 & \text{si } 0 < score < 2 \end{cases}$$

Estas fórmulas entregan reglas de parámetros como se muestran en la Figura 4.2. Se utilizaron los límites indicados con el fin de permitir más movimientos a los usuarios con mayor puntaje, llegando a 10[°] y 10[°/s] como máximo, y a 5[°] y 3[°/s] como mínimo, en los ángulos *pitch* y *roll*. Al desactivar el asistente, el valor máximo para los ángulos es de 15[°].

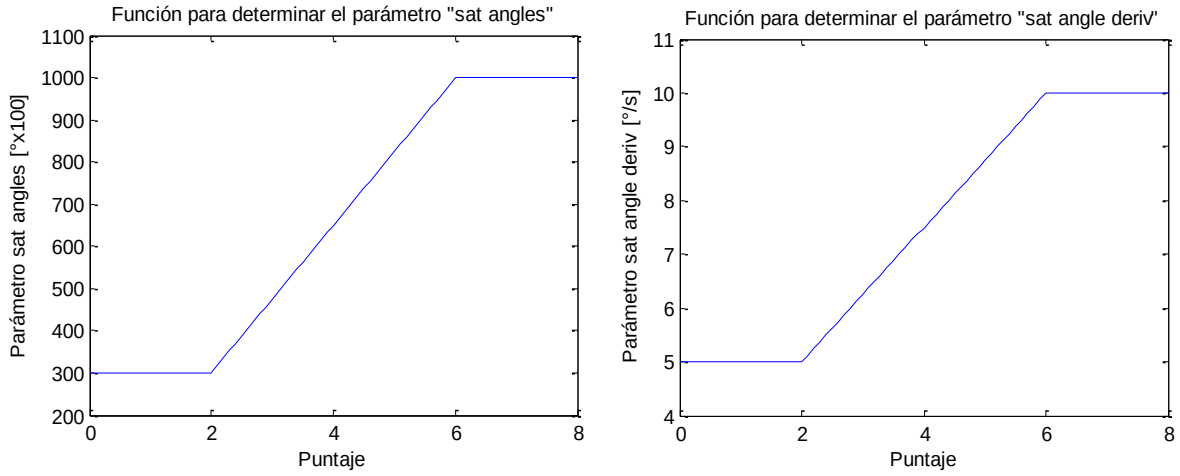


Figura 4.2: Gráficos de la determinación de los parámetros 'sat_angles' (izq.) y 'sat_angle_deriv' (der.) a partir del puntaje ponderado total obtenido en la prueba de desempeño.

Como caso límite, si el puntaje ponderado obtenido es de 0, eso significa que el operario falló completamente en todas las pruebas y no es capaz de volar un quad-rotor sin práctica previa, por lo que se le sugiere el uso de un simulador para adquirir práctica.

Con estas funciones para definir los parámetros, se decidió utilizar los siguientes ponderadores:

- Pitch-Roll: $w_{pr} = 0.6$
- Aceleración: $w_{th} = 0.4$

- Posicionamiento: $w_{GPS} = 0$

Estos pesos entregaron puntajes ponderados para las pruebas de los cuatro operarios que se indican en la Tabla 4.6.

Tabla 4.6: Puntajes obtenidos para todos los operarios utilizando los ponderadores y valores de los parámetros calculados.

Operario	*Pitch-Roll*	Aceleración	Puntaje Ponderado	sat_angles	sat_angle_deriv
1	9,2	9,2	9,2	-	-
2	4,9	2,6	4,0	641,25	7,44
3	4,2	4,1	4,2	678,88	7,71
4	4,2	4,0	4,1	662,25	7,59

4.3. Prueba con el sistema de control asistido activado

4.3.1. RESULTADOS

En el Anexo XIX (pág. 98) se muestran los resultados de la misma prueba realizada con el sistema de control asistido activado para los operarios 2, 3 y 4.

4.3.2. ANÁLISIS COMPARATIVO

Se puede ver que diferentes operarios muestran niveles de habilidad distintos en las competencias requeridas para el vuelo de UAVs. El primer operario muestra un control total de la nave, pudiendo ceñirse a los requerimientos sin errores, lo que era esperable debido a su nivel de experiencia previa.

Por otro lado, el segundo operario presentó un manejo que muestra poca experticia, al tener dificultades elevando la nave así como al controlar los ángulos. En la Figura 4.3 se observa este comportamiento, llegando a valores de hasta 10[°] en el ángulo *roll*, y pudiéndose mantener no más de 10 [s] por sobre la altitud de 1[m]. El control de los ángulos mejoró ligeramente al utilizar el asistente, disminuyendo las oscilaciones. Sin embargo, el usuario mostró dificultades para adaptarse al control modificado. En la última prueba realizada (Figura 4.3) logró elevar la nave y estabilizarla durante 15 [s].

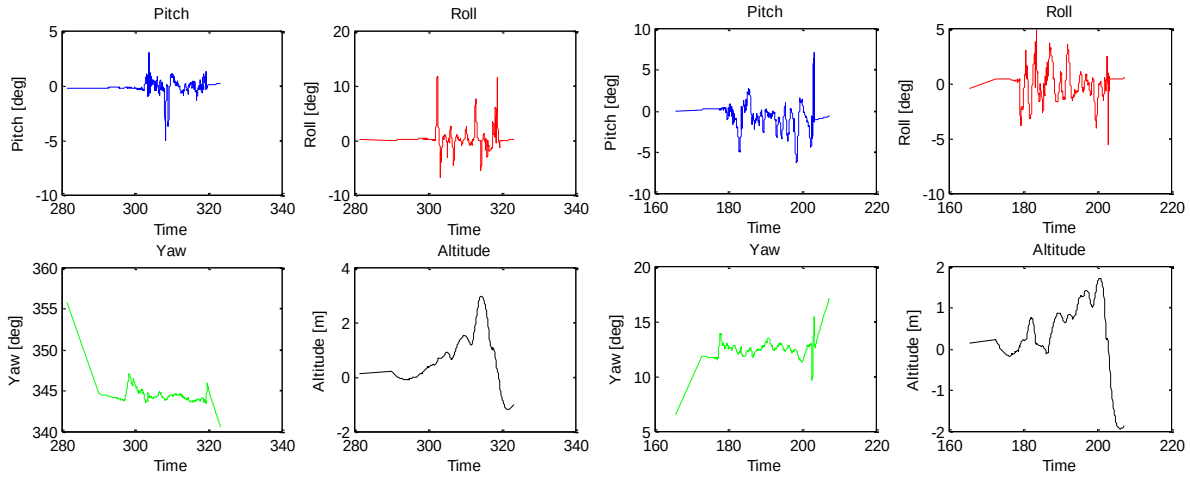


Figura 4.3: Prueba del segundo operario sin el asistente (izq.) y con el asistente (der.). Se destaca el corto tiempo que logra mantener la nave por sobre el umbral de 1[m].

El tercer operario mostró dificultades en aterrizaje y control de aceleración, lo que se observa en algunas pruebas en las curvas de altitud. Sin embargo, logró adaptarse al control asistido de manera más rápida

que el segundo operario. En la Figura 4.4 se muestran las respuestas tanto con el asistente desactivado como activado, las cuales presentan una diferencia tanto en las oscilaciones angulares obtenidas, así como la estabilidad en altitud alcanzada.

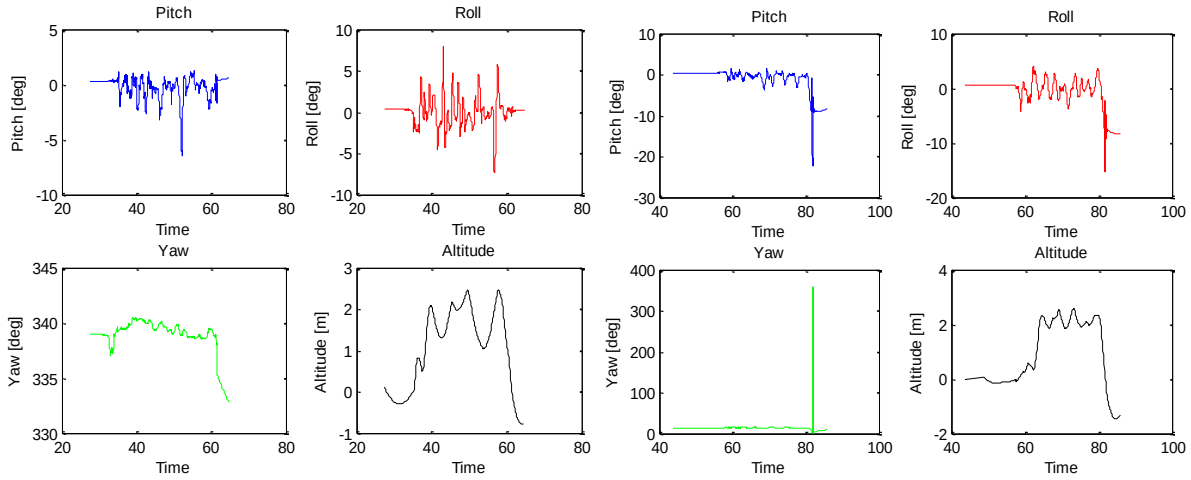


Figura 4.4: Pruebas del tercer operario sin asistente (izq.) y con asistente (der.). Se pueden observar oscilaciones lentas de menor amplitud en los ángulos *pitch* y *roll* al utilizar el asistente, y una altitud más estable durante la prueba.

El cuarto operario presenta un manejo regular de la nave, pudiendo elevarla y controlarla con oscilaciones, las cuales se reducen al implementar el asistente como se muestra en la Figura 4.5. Luego de varias pruebas, éste pudo controlar con mayor precisión la altitud manteniendo una postura estable.

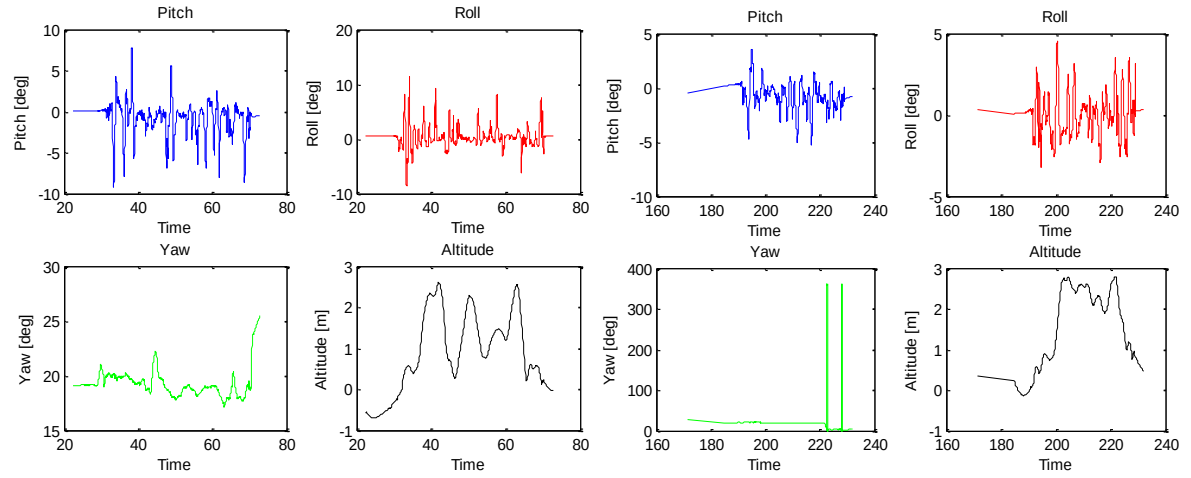


Figura 4.5: Pruebas del cuarto operario sin asistente (izq.) y con asistente (der.). Se puede observar la diferencia en el control de altitud, además de los valores reducidos en los ángulos pitch y roll.

A modo de análisis general, se puede observar en las curvas de respuesta que los usuarios tratan de sobre-compensar la diferencia en la respuesta de los controles al tener el asistente encendido, lo que se traduce en oscilaciones lentas en los ángulos *pitch* y *roll*. Sin embargo, dichos valores se mantienen acotados por las limitaciones impuestas por el sistema de control asistido, a diferencia de las pruebas originales en donde usuarios inexpertos llegan a valores de más de 10° en ambos ángulos. Los valores limitados aseguran la estabilidad aérea de la nave. En efecto, los operarios lograron completar las pruebas manteniendo mayor estabilidad en altitud, lo que refleja el efecto del asistente en la operación realizada por los usuarios, los cuales logran mayor concentración en controlar la altitud de la nave en lugar de la estabilidad angular.

Por otro lado, a modo de análisis cualitativo, los usuarios declararon haber experimentado un cambio positivo en el control general de la nave al activar el asistente, facilitando su experiencia de manejo. El segundo operario mostró dificultades para adaptarse al cambio, excediendo la compensación en las referencias angulares, lo que llevó a tener más oscilaciones lentas.

Capítulo 5: Conclusiones y Trabajo Futuro

En este trabajo se logró cumplir gran parte de los objetivos planteados inicialmente. Particularmente, se considera que:

- Se logró comprender de manera general el funcionamiento de las aeronaves no tripuladas, y de manera más específica tanto las leyes que rigen el comportamiento de los quad-rotors, así como los diferentes tipos de controladores que existen para controlar su comportamiento.
- Se realizó un estudio exhaustivo de los componentes que son necesarios para la integración completa de un quad-rotor, con todos los módulos requeridos.
- Se estudió el escenario actual a nivel global en el que se integran las aeronaves no tripuladas, comprendiendo el estado del arte en el que se encuentra dicha tecnología.
- Se logró ensamblar, conectar y poner en marcha una aeronave en base a *firmware* de código abierto, cuyo desarrollo está destinado tanto para uso personal como para fines investigativos. Se determinó además el grado de utilidad de dichos recursos, estableciéndose sus limitaciones.
- Se diseñó un sistema de control asistido desde un punto de vista del aprendizaje del usuario. El sistema diseñado consideró tanto factores técnicos del manejo de aeronaves, así como el factor humano del aprendizaje.
- Se implementó el sistema de control asistido en forma de un *firmware* modificado, incluyendo un modo de vuelo diseñado específicamente para su uso asistiendo a usuarios con dificultades.
- Se diseñó una prueba de desempeño desde un punto de vista funcional, en conjunto con un programa para analizarla y calificarla de manera acertada.
- Se incluyó el uso de un simulador en la prueba para extender su alcance, aun cuando éste no logró mostrar la precisión deseada. Se logró además integrar el simulador tanto con la prueba real así como con el análisis *a posteriori*.
- Se logró evaluar el asistente a partir de pruebas realizadas con diferentes usuarios, validando el alcance del sistema completo, incluyendo la prueba, el análisis de desempeño y el sistema de notas.

Se considera que, de manera general, los resultados presentan aspectos positivos y negativos. Específicamente, el sistema de control asistido mostró ser funcional, logrando ayudar a la estabilización de la nave. Se observó que la estabilización en el plano, si bien es un aspecto importante, no abarca todos los elementos que deben aprenderse mediante la experiencia, siendo el otro el buen manejo de la aceleración vertical. Para poder trabajar con esa variable e intervenir su control, es necesario tener mayor precisión en las mediciones de altitud. Esto se podría lograr tanto con un sistema de posicionamiento externo (flujo óptico, posicionamiento 3D mediante cámaras), así como mediante el uso de un sensor de altura más preciso (sonar, láser). Si bien se contaba con un sensor sonar, su integración con el sistema APM: Copter no logró ser realizada en el marco de este trabajo ya que presentó ser de una carga de trabajo considerable, por lo que resultaría interesante un replanteamiento del problema a partir de lo aquí discutido y con herramientas adicionales.

Por otro lado, se esperaba poder obtener mejores resultados simulados de posicionamiento. El uso de un simulador compatible previamente con el código fuente simplificó la tarea del uso de parámetros, sin embargo mostró no ser preciso en las simulaciones. En efecto, se intentó realizar otros tipos de pruebas (desplazamientos en los ejes X e Y) además de la prueba de desempeño que sí se efectuó. Sin embargo, no se pudo obtener una medición consistente con los desplazamientos, lo que descartó esa posibilidad. Nuevamente, un sistema más sofisticado de sensores podría permitir utilizar estas funciones de una manera más provechosa.

A partir de los resultados expuestos en este trabajo, se puede concluir que el asistente de vuelo implementado representa una mejora al proceso de aprendizaje de operación de una aeronave, en donde puede intervenir para facilitar el manejo de naves a operarios con poca práctica. Debido a que el sistema está basado en pruebas de desempeño, no es factible su uso con operarios sin experiencia alguna. En ese

caso, se puede comenzar por el uso de un simulador para adquirir práctica básica y volver a intentarlo. Por otro lado, también se destaca que el efectivo funcionamiento del sistema de control asistido se enmarca en un proceso iterativo de prueba y error, el cual no puede ser reemplazado por conocimientos teóricos. Esta etapa puede tener mayor o menor duración dependiendo del operario, ya que cada persona presenta diferentes tasas de aprendizaje de manera natural según sus habilidades.

Este sistema puede ser aplicado a otro tipo de naves, ya sea de una manera análoga, con pruebas de desempeño y un análisis *off-line*, o bien de manera más automatizada, incluyendo algún tipo de análisis en línea que modifique los parámetros requeridos de manera automática, evitando así tener que realizar procesamiento de datos fuera de línea. También se puede considerar el uso de coordenadas GPS para la obtención de estimaciones más precisas de posicionamiento en 3D, sin embargo esto restringe las condiciones de operación.

Glosario y Acrónimos

APM:	ArduPilotMega. Corresponde a una plataforma de vuelo de UAVs capaz de controlar multicópteros y aviones. Es desarrollado por la comunidad "DIY Drones" y es de código abierto.
BLDC:	*Brushless DC* – Motor de corriente continua sin escobillas.
DIY:	*Do it yourself* – Hágalo usted mismo, término utilizado para denominar procedimientos que el usuario puede realizar por su cuenta, contando con las herramientas adecuadas. Generalmente se refiere a proyectos de construcción o ensamblaje.
ESC:	*Electronic Speed Controller* – Controlador electrónico de velocidad. Unidad que utiliza una señal PWM de entrada para entregar una señal trifásica de poder a un motor BLDC.
Firmware:	Conjunto de datos almacenados en sistemas electrónicos que agrupan programas e información en memoria de lectura y escritura (EEPROM, Flash), dictando la lógica de bajo nivel que controla dicho sistema.
Frame:	Estructura principal que sirve de esqueleto de un UAV, en donde se montan todos los módulos necesarios para su funcionamiento.
GPS:	*Global positioning system* – Sistema de posicionamiento global. Método moderno de ubicación georeferencial, en la cual se utilizan señales combinadas de diversos satélites para estimar una posición en el mapa. La precisión del método depende de la disponibilidad y el alcance de la señal de los satélites. Se debe utilizar un módulo dedicado para detectar dichas señales.
IMU:	*Inertial measurement unit* – Unidad de medición inercial. Término para denominar el módulo que combina mediciones de acelerómetros y giróscopos para poder estimar una postura instantánea en base a velocidades y aceleraciones angulares, sin depender de instrumentos externos.
I/O:	*Input/Output* – Entrada/salida.
MAVLink:	*Micro-Air-Vehicle Link* - Protocolo de comunicación desarrollado específicamente para transmisión de información con UAVs.
MIMO:	*Multiple-Input Multiple-Output* –Tipo de sistema que consta de más de una variable de entrada y más de una variable de salida.
Open-source:	Código abierto. Se utiliza para denominar proyectos en los cuales la información puede ser usada, estudiada, modificada y compartida por los usuarios de manera gratuita.
PID:	Proporcional-Integral-Derivativo – Tipo de controlador que combina los efectos de tres controladores más simples.
Pitch:	Ángulo de elevación.
PPM:	*Pulse Position Modulation* – Modulación por posición de pulso. Método de modulación digital de señales que codifica la información de diversas señales (por ejemplo PWM) en una sola señal electrónica.
PWM:	*Pulse Width Modulation* – Modulación por ancho de pulso. Método de modulación electrónica de señales que transforma una señal moduladora en el ancho de una señal temporal (duración de un pulso) para su transmisión entre dispositivos. Se utiliza normalmente para enviar señales de control desde módulos de bajo voltaje hacia unidades de control de motores, y también para comunicación inalámbrica.
Quad-rotor:	Tipo de UAV que consta de 4 rotores dispuestos de manera que propulsan la nave para elevarla. También es llamado quadricóptero o quadrirotor.
Roll:	Ángulo de alabeo.
SITL:	*Software-in-the-loop* – Sistema simulador de *firmware* desarrollado por la comunidad de APM, que posibilita la realización de vuelos simulados ejecutando el código original.
UAV:	*Unmanned Aerial Vehicle* – Vehículo aéreo no tripulado, también llamado dron.
Yaw:	Ángulo de dirección.

Bibliografía

[1] A. Dzul y R. Lozano P. Castillo, *Modelling and Control of Mini-Flying Machines*. New York: Springer-Verlag, 2005.

[2] APM Copter. (2013, Octubre) ArduPilot - What you'll need. [Online]. http://copter.ardupilot.com/wiki/introduction/what-you-need/

[3] Robert Beatty. (2011, Octubre) DIYDrones. [Online]. http://diydrones.com/profiles/blogs/quadrotor-trying-a-few-design-ideas

[4] 3DRobotics. (2013, Octubre) 3DR ArduCopter Quad C Frame Kit. [Online]. http://store.3drobotics.com/products/3dr-arducopter-quad-c-frame-kit-2

[5] I Gonzalez, S Salazar, and H Romero, "Attitude control of a quad-rotor using speed sensing in brushless DC motors," in *2011 8th International Conference on Electrical Engineering Computing Science and Automatic Control (CCE)*, Merida City, 2011, pp. 1-6.

[6] APC Propellers. (2008) Materials Research. [Online]. http://www.apcprop.com/v/Research/research.html#materials

[7] 3DRobotics. (2013) A2830 Out Runner Brushless Motor Instruction. Datasheet.

[8] QuietFlyer Magazine. (2013) How Electric Motors Work. [Online]. http://www.stefanv.com/rcstuff/qf200212.html

[9] S Jeremia, E Kuantama, and J Pangaribuan, "Design and control of Remote-controlled quad-copter based on STC12C5624AD," in *2012 International Conference on System Engineering and Technology (ICSET)*, Bandung, 2012, pp. 1-6.

[10] 3DRobotics. (2013) PX4 FMU (Flight Management Unit) + IO (Input/Output) Kit. [Online]. http://store.3drobotics.com/products/px4-fmu-flight-management-unit-plus-io-input-slash-output-kit

[11] 3DRobotics. (2013) 3DR Pixhawk. [Online]. http://store.3drobotics.com/products/3dr-pixhawk

[12] Starlino. (2009) A guide to using IMU (Accelerometer and Gyroscope Devices) in embedded applications. [Online]. http://www.starlino.com/imu_guide.html

[13] E Altug, J Ostrowsky, and R Mahony, "Control of a Quadrotor Helicopter Using Visual Feedback," in *Proceedings of the 2002 IEEE International Conference on Robotics &Automation*, Washington DC, 2002, pp. 72-77.

[14] E Altug and C Taylor, "Vision-based pose estimation and control of a model helicopter," in *Proceedings of the IEEE International Conference on Mechatronics, 2004. ICM '04*,

2004, pp. 316 - 321.

[15] Multicopter Wiki. (Agosto, 2013) Multicopter Projects. [Online]. http://multicopter.org/wiki/Multicopter_Table

[16] APM: Mission Planner. Mission Planner Features. [Online]. http://planner.ardupilot.com/wiki/mission-planner-features/

[17] Chris Anderson. (2014) DIY Drones. [Online]. http://www.diydrones.com/

[18] 3d Robotics. (2013) 3DRobotics Store. [Online]. http://store.3drobotics.com

[19] MultiWii. MultiWii Forum. [Online]. http://www.multiwii.com/forum/

[20] OpenPilot. (2013) Community. [Online]. http://www.openpilot.org/community/

[21] OpenPilot. OpenPilot Store. [Online]. http://store.openpilot.org/

[22] APM: Copter. Flight Modes. [Online]. http://copter.ardupilot.com/wiki/flight-modes/

[23] Leonard Thall. (2013, Enero) ArduCopter 2.9 PID Loops for STABILIZE, ACRO and ALT_HOLD. [Online]. http://diydrones.com/profiles/blogs/arducopter-2-9-pid-loops-for-stabilize-acro-and-alt-hold

[24] J.P. How, "Increasing autonomy of UAVs," *Robotics & Automation Magazine, IEEE*, vol. 16.2, pp. 43-51, 2009.

[25] B Pattipati, C Sankavaram, and K.R. Pattipati, "System identification and Estimation Framework for pivotal automotive battery management system characeristics," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 41, no. 6, pp. 869-884, Enero 2011.

[26] B Saha, C Koshimoo, and C.C Quach, "Battery health management system for electric UAVS," in *Aerospace Conference, 2011 IEEE* , Big Sky, MT, 2011, pp. 1-9.

[27] APM. (2013, Diciembre) Mission Planner Download. [Online]. http://ardupilot.com/downloads/?did=82

[28] APM: Copter. (2014, Junio) Tuning. [Online]. http://copter.ardupilot.com/wiki/tuning/

[29] Dave C. (2012, Noviembre) Arducopter Tuning Guide - DIY Drones. [Online]. http://diydrones.com/forum/topics/arducopter-tuning-guide

[30] APM: Copter. (2014, Junio) Diagnosing problems using logs. [Online]. http://copter.ardupilot.com/wiki/common-diagnosing-problems-using-logs/

[31] APM: Copter. First Flight. [Online]. http://copter.ardupilot.com/wiki/flying-arducopter/

[32] Pixhawk.org. (Agosto, 2014) PX4Flow Smart Camera. [Online]. http://www.pixhawk.org/modules/px4flow

[33] APM: Copter. (2014, Mayo) SITL: Software in the Loop Simulator. [Online]. http://copter.ardupilot.com/wiki/common-sitl-software-in-the-loop-simulator/

[34] Andrew Tridgell. (2014, Mayo) MAVProxy. [Online]. http://tridge.github.io/MAVProxy/

[35] Noah Spurrier. (2008) Python: module pexpect. [Online]. http://pexpect.sourceforge.net/pexpect.html

[36] Andrew Tridgell. (2011, Diciembre) GitHub: mavutilpy in pymavlink. [Online]. https://github.com/mavlink/pymavlink/blob/master/mavutil.py

[37] Andrew Tridgell. (2011, Agosto) API Documentation for PyMAVLink. [Online]. http://www.samba.org/tridge/UAV/pymavlink/apidocs/index.html

[38] P Osborne. (2013, Septiembre) The Mercator Projections. [Online]. http://www.mercator99.webspace.virginmedia.com/mercator.pdf

[39] DIYDrones. (2014, Marzo) GitHub - diydrones/ardupilot. [Online]. https://github.com/diydrones/ardupilot

[40] Fernandez, F. DIYDrones. (2014, Junio) Ardupilot, GitHub. [Online]. https://github.com/Merkoa/ardupilot

Anexos

Anexo I - Programa **rw_rcin.m**

```
% Script to rewrite RC inputs into several files with new format

clear all
close all
clc

user = input('Nombre operario:\n','s');

i = 1;
while(true)
    try
        load(fullfile(user,strcat('data_dflog',num2str(i),'.mat')));
    catch err
        if(i == 1)
            error(strcat('no data found for ',user,' user'));
        else
            break;
        end
    end

    % Extraction of RC commands from logs into cell arrays
    time_signal = RCIN(:,2);

    RC1 = split_signal(RCIN(:,3),time_signal);
    RC2 = split_signal(RCIN(:,4),time_signal);
    RC3 = split_signal(RCIN(:,5),time_signal);
    RC4 = split_signal(RCIN(:,6),time_signal);
    n(i) = length(RC1);

    RC{i} = cell(4,n(i));
    RC{i}(1,:) = RC1;
    RC{i}(2,:) = RC2;
    RC{i}(3,:) = RC3;
    RC{i}(4,:) = RC4;

    i = i + 1;
end
% Writing formated RC inputs into new files for simulation tests
L = i - 1;


for i = 1:L
    for j = 1:n(i)
        k = j + sum(n(1:(i-1)));
        filename = strcat(user,'\',sprintf('rcin_test%d.txt',k));
        file = fopen(filename,'w');
        for t = 1:length(RC{i}{1,j})
            for c = 1:4
                fprintf(file,'rc %d %d\n',c,RC{i}{c,j}(t));
            end
            fprintf(file,'sleep\n');
        end
```

```
        fclose(file);
    end
end
```

Anexo II - Programa **sim_arducopter_ff.sh**

```
#!/bin/bash

# terminate processes
killall -q ArduCopter.elf
pkill -f sim_multicopter.py

# convert original APM log information into txt file with RC PWM data
# rw_rcin.sh log.log
sleep 2

# compile aircraft simulator
set -e
set -x

target=sitl
frame="X"

echo "Building with target $target for frame $frame"

autotest=$(dirname $(readlink -e $0))
pushd $autotest/../../ArduCopter
make clean $target

# execute simulator
tfile=$(mktemp)
echo r > $tfile
#gnome-terminal -e "gdb -x $tfile --args /tmp/ArduCopter.build/ArduCopter.elf"
gnome-terminal -e /tmp/ArduCopter.build/ArduCopter.elf
#gnome-terminal -e "valgrind -q /tmp/ArduCopter.build/ArduCopter.elf"
sleep 2
rm -f $tfile
gnome-terminal -e "../Tools/autotest/pysim/sim_multicopter.py --frame=$frame --home=-
33.457893,-70.662041,584,270"
sleep 2
popd

mavtest3.py

sleep 5
echo "fin del programa"
pkill -f sim_multicopter.py
killall -q ArduCopter.elf
```

Anexo III - Programa **mavsim_pexpect.py**

```
#! /usr/bin/env python

#
#   mavsim_pexpect.py
#
#   Written by:
#   Felipe Fernandez
#   Universidad de Chile
#
#   uses RC log data from real flight to simulate it using SITL and MAVProxy
#   this script uses pexpect to send commands to MAVProxy
#   more info: http://copter.ardupilot.com/wiki/common-sitl-software-in-the-loop-simulator/
#   other references:
#   Undergraduate Thesis: 'Diseno e implementacion de un sistma de contron asistido para plataforma aerea multi-rotor'
#   (Universidad de Chile)
#

import sys, os
import pexpect

from pymavlink import mavutil
import time

cmd = 'mavproxy.py --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --out 127.0.0.1:14550 --out 127.0.0.1:14551 --out 192.168.231.1:14550 --console --map --aircraft test --quadcopter $*'

# start MAVProxy via pexpect spawn command

mavproxy = pexpect.spawn(cmd, logfile=sys.stdout, timeout = 60)
mavproxy.delaybeforesend = 0

# create a mavlink instance as a UDP port connection
try:
    mav = mavutil.mavlink_connection('127.0.0.1:14551', robust_parsing=True)
except Exception, msg:
    print("Failed to start mavlink connection on 127.0.0.1:14551" % msg)
    raise

mavproxy.expect('Saved')

# import saved parameters
mavproxy.send('param load parametros.param\n')
mavproxy.expect('Loaded')

# arm motors
print('Armando motores')
mavproxy.send('mode STABILIZE')
mavproxy.send('arm uncheck all\n')
mavproxy.send('arm throttle\n')
mav.motors_armed_wait()

# make sure we have GPS fix
```

```
mav.wait_gps_fix()

# file reading loop
log_rc = open('rcin.txt','r')
freq = 50.0 #log reading frequency in hertz, must be same as RC log freq
time_delay = 1/freq
line_count = 0
t_final = 60 #simulation time in seconds

# time counter
time_0 = time.time()
for line in log_rc:
    if (line.rstrip('\n')) == 'sleep':
        # here we must calculate how much time to wait to have a sleep time as close
as possible to $time_delay (because of processing time)
        end_time = start_time + time_delay
        sleep_time = end_time - time.time()
        time.sleep(sleep_time)

        line_count = line_count + 1
    else:
        if 'rc 1' in line:
            start_time = time.time()
        mavproxy.send(line)
    # uncomment these 2 lines to stop simulation at $t_final
    if line_count>= t_final*freq:
        break
print('Tiempo total: ' + str(time.time()-time_0))
# close file and pexpect instance

log_rc.close()
mavproxy.terminate()
```

Anexo IV - Programa **mavsim_udp.py**

```
#! /usr/bin/env python

#
#   mavsim_udp.py
#
#   Written by:
#   Felipe Fernandez
#   Universidad de Chile
#
#   uses RC log data from real flight to simulate it using SITL and MAVProxy
#   this script uses the mavutil library to send commands to MAVProxy via a UDP port
connection
#   more info: http://copter.ardupilot.com/wiki/common-sitl-software-in-the-loop-
simulator/
#   other references:
#   Undergraduate Thesis: 'Diseno e implementacion de un sistma de contron asistido
para plataforma aerea multi-rotor'
#   (Universidad de Chile)
#

import sys, os
import pexpect

from pymavlink import mavutil
import time

# utility function to send RC commands to SITL via connection named "udp_port",
# defined as a UDP port connection, in this case 14551, which must be indicated
# with the --out command in the string 'cmd' used by pexpect

def set_attitude(rc1, rc2, rc3, rc4):
    global udp_port
    values = [ 65535 ] * 8
    values[0] = rc1
    values[1] = rc2
    values[2] = rc3
    values[3] = rc4
    udp_port.mav.rc_channels_override_send(udp_port.target_system,
udp_port.target_component, *values)

cmd = 'mavproxy.py --master tcp:127.0.0.1:5760 --sitl 127.0.0.1:5501 --out
127.0.0.1:14550 --out 127.0.0.1:14551 --out 192.168.231.1:14550 --console --map --
aircraft test --quadcopter $*'

# start MAVProxy via pexpect spawn command

mavproxy = pexpect.spawn(cmd, logfile=sys.stdout, timeout = 60)
mavproxy.delaybeforesend = 0

# create a mavlink instance as a UDP port connection

udp_port = mavutil.mavlink_connection('udp:127.0.0.1:14551')

mavproxy.expect('Received [0-9]+ parameters')
```

```
# import saved parameters
mavproxy.send('param load parametros.param\n')
mavproxy.expect('Loaded')
time.sleep(1)

# arm motors
udp_port.wait_heartbeat()
udp_port.set_mode('STABILIZE')
print('\nArmando motores')

mavproxy.send('arm uncheck all\n')
time.sleep(1)
udp_port.arducopter_arm()
udp_port.motors_armed_wait()

# make sure we have GPS fix
udp_port.wait_gps_fix()

# file reading loop
log_rc = open('rcin.txt','r')
freq = 50.0 #log reading frequency in hertz, must be same as RC log freq
time_delay = 1/freq
line_count = 0
t_final = 60 #simulation time in seconds

# time counter
time_0 = time.time()
for line in log_rc:
    if (line.rstrip('\n')) == 'sleep':
        # here we must calculate how much time to wait to have a sleep time as close
as possible to $time_delay (because of processing time)
        end_time = start_time + time_delay
        sleep_time = end_time - time.time()
        time.sleep(sleep_time)

        line_count = line_count + 1
    else:
        # we must identify the rc num and assign the data
        [txt,num,val] = line.split(' ')
        if  int(num) == 1:
            # reset time for the wait loop
            start_time = time.time()
            rc1 = int(val)
        elif int(num) == 2:
            rc2 = int(val)
        elif int(num) == 3:
            rc3 = int(val)
        elif int(num) == 4:
            rc4 = int(val)
            # when we have all 4 PWM values for the rc, we send it via udp
            set_attitude(rc1, rc2, rc3, rc4)
    # uncomment these 2 lines to stop simulation at $t_final
    #if line_count>= t_final*freq:
    #   break
print('Tiempo total: ' + str(time.time()-time_0))
# close file and pexpect instance
```

```
log_rc.close()
mavproxy.terminate()
```

Anexo V - Programa **analyze_logs.m**

```
%   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   %%%%%%%%%%  LOG ANALYZING SCRIPT %%%%%%%%%%%%%%%
%   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% analyze_logs.m
% Written by Felipe Fernandez
% Universidad de Chile
%
% processes logs both from real and simulated data to deliver a performance
% metric value reflecting the flying operator's skill and dexterity
%
% needs data set from all flight tests, which have to be under the folder
% 'username'\'sim' or 'real' for each set of logs. simulated data is
% converted directly from telemetry logs (.tlog), and real data is
% converted direclty from dataflash logs (.log) using Mission Planner.
%
% log name format:
%       'username'\real\log_realX.mat
%       'username'\sim\flightX\flight.tlog.mat
%   where X is the test no.
%
% reference:
%   Undergraduate Thesis: 'Dise Implementaci e un sistema de control
%   asistido para plataforma a a multi-rotor'
%   2014, Universidad de Chile
%

close all
clear all
clc

%% Pre-processing

% File read of number of tests
user = input('Nombre operario:\n','s');
bool = input('Usuario utiliz istente? 1 = si, 0 = no\n');

split_logs(user);

list = dir(fullfile(user,'real','log_real*'));
last = list(end).name;
last_2 = strrep(last,'log_real','');
last_3 = strrep(last_2,'.mat','');
N = str2double(last_3);

% List log names for simulated and real tests
log_sim = cell(N,1);
for i = 1:N
    log_sim{i} = fullfile(user,'sim',strcat('flight',num2str(i)),...
        'flight.tlog.mat');
end

log_real = cell(N,1);
for i = 1:N
    log_real{i} = fullfile(user,'real',strcat('log_real',num2str(i),'.mat'));
```

```
end

save('data_names.mat','log_sim','log_real','N','user','bool')

pitch_real = cell(1,N);
roll_real = cell(1,N);
yaw_real = cell(1,N);
alt_real = cell(1,N);
climb_real = cell(1,N);
time_att_real = cell(1,N);
time_alt_real = cell(1,N);

lat_sim = cell(1,N);
lon_sim = cell(1,N);
gps_time_sim = cell(1,N);

save('data_real.mat','pitch_real','roll_real','yaw_real','alt_real',...
        'climb_real','time_att_real','time_alt_real');
save('data_sim.mat','lat_sim','lon_sim','gps_time_sim');

clear all
%% Loading of real and simulated data

% Loading real flight pose data
load('data_names.mat')

for i = 1:N
    load('data_names.mat')
    load('data_real.mat')

    load(log_real{i});

    % attitude in degrees
    pitch_real{i} = pitch_log;
    roll_real{i} = roll_log;
    yaw_real{i} = yaw_log;

    % system time from attitude
    time_att_real{i} = time_att_log;

    % altitude and climb rate
    alt_real{i} = alt_log;% - min(alt_log);

    climb_real{i} = climb_log;

    time_alt_real{i} = time_alt_log;

    % No GPS data available

    save('data_real.mat','pitch_real','roll_real','yaw_real','alt_real',...
        'climb_real','time_att_real','time_alt_real');

    if(~bool)
        % Loading simulated data
        load('data_sim.mat')

        load(log_sim{i});
```

```
        % GPS coordinates
        lat_sim{i} = lat_mavlink_global_position_int_t(:,2);
        lon_sim{i} = lon_mavlink_global_position_int_t(:,2);

        % conversion to deg units
        lat_sim{i} = lat_sim{i}/10^7;
        lon_sim{i} = lon_sim{i}/10^7;

        % saving time arrays from simulation
        gps_time_sim{i} = time_boot_ms_mavlink_global_position_int_t;

        % NOT USED
        % conversion from GEO to degrees, minutes, seconds
        % lat_deg = fix(lat_sim);
        % lat_min = mod(fix(abs(lat_sim)*60),60);
        % lat_sec = mod(abs(lat_sim)*3600,60);
        %
        % lon_deg = fix(lon_sim);
        % lon_min = mod(fix(abs(lon_sim)*60),60);
        % lon_sec = mod(abs(lon_sim)*3600,60);

        % save data

        save('data_sim.mat','lat_sim','lon_sim','gps_time_sim');
    end

    clear all
end


%% Data processing

% reload data
load data_names.mat
load data_real.mat
if(~bool)
    load data_sim.mat


    %{
    % decimal degrees scaling to meters using the approximate length of a
    % degree of latitude and longitude according to WGS84 model of the earth
    % we assume that all the simulations were run with the same virtual origin
    lat_mean = mean(lat_sim{1});

    l_lat = length_latdeg(lat_mean);
    l_lon = length_londeg(lat_mean);

    % GPS degree scaling and unbiasing with length of degree and initial
    % position as origin
    % we obtain position in earth's X-Y plane, in meters
    lat_scaled = cell(1,N);
    lon_scaled = cell(1,N);

    for i = 1:N
        lat_scaled{i} = (lat_sim{i} - lat_sim{i}(1))*l_lat;
```

```
        lon_scaled{i} = (lon_sim{i} - lon_sim{i}(1))*l_lon;
    end

    %}

    %%{
    % using UTM transformation
    % gettnig current zone
    p = [lat_sim{1}(1),lon_sim{1}(1)];
    zone = utmzone(p);

    % constructing the utmsctruct object
    [ell,estr] = utmgeoid(zone);

    utmstruct = defaultm('utm');
    utmstruct.zone = zone;
    utmstruct.geoid = ell;
    utmstruct = defaultm(utmstruct);

    % transformating obtained lat and lon values to projected grid coordinates
    lat_scaled = cell(1,N);
    lon_scaled = cell(1,N);

    for i = 1:N
        L = length(lat_sim{i});
        lat_scaled{i} = zeros(1,L);
        lon_scaled{i} = zeros(1,L);

        for j = 1:L
            [lat_scaled{i}(j),lon_scaled{i}(j)] = ...
                mfwdtran(utmstruct,lat_sim{i}(j),lon_sim{i}(j));
        end
        lat_scaled{i} = lat_scaled{i} - lat_scaled{i}(1);
        lon_scaled{i} = lon_scaled{i} - lon_scaled{i}(1);
    end
    %}
end

% Calculation of basic signal information
pitch_mean = zeros(1,N);
pitch_std = zeros(1,N);
pitch_min = zeros(1,N);
pitch_max = zeros(1,N);

roll_mean = zeros(1,N);
roll_std = zeros(1,N);
roll_min = zeros(1,N);
roll_max = zeros(1,N);

yaw_mean = zeros(1,N);
yaw_std = zeros(1,N);
yaw_min = zeros(1,N);
yaw_max = zeros(1,N);

alt_mean = zeros(1,N);
alt_std = zeros(1,N);
```

```
alt_min = zeros(1,N);
alt_max = zeros(1,N);

climb_mean = zeros(1,N);
climb_std = zeros(1,N);
climb_min = zeros(1,N);
climb_max = zeros(1,N);

if(~bool)
    lat_mean = zeros(1,N);
    lat_std = zeros(1,N);
    lat_min = zeros(1,N);
    lat_max = zeros(1,N);

    lon_mean = zeros(1,N);
    lon_std = zeros(1,N);
    lon_min = zeros(1,N);
    lon_max = zeros(1,N);
end

for i = 1:N
    % Mean and standard deviation
    pitch_mean(i) = mean(pitch_real{i});
    pitch_std(i) = std(pitch_real{i});

    roll_mean(i) = mean(roll_real{i});
    roll_std(i) = std(roll_real{i});

    yaw_mean(i) = mean(yaw_real{i});
    yaw_std(i) = std(yaw_real{i});

    alt_mean(i) = mean(alt_real{i});
    alt_std(i) = std(alt_real{i});

    climb_mean(i) = mean(climb_real{i});
    climb_std(i) = std(climb_real{i});

    if(~bool)
        lat_mean(i) = mean(lat_scaled{i});
        lat_std(i) = std(lat_scaled{i});

        lon_mean(i) = mean(lon_scaled{i});
        lon_std(i) = std(lon_scaled{i});
    end

    % Min - max
    pitch_min(i) = min(pitch_real{i});
    pitch_max(i) = max(pitch_real{i});

    roll_min(i) = min(roll_real{i});
    roll_max(i) = max(roll_real{i});

    yaw_min(i) = min(yaw_real{i});
    yaw_max(i) = max(yaw_real{i});

    alt_min(i) = min(alt_real{i});
    alt_max(i) = max(alt_real{i});
```

```
    climb_min(i) = min(climb_real{i});
    climb_max(i) = max(climb_real{i});

    if(~bool)
        lat_min(i) = min(lat_scaled{i});
        lat_max(i) = max(lat_scaled{i});

        lon_min(i) = min(lon_scaled{i});
        lon_max(i) = max(lon_scaled{i});
    end
end

%% Test analysis
% a total of 3 scores for each test:
% 1 = pitch-roll stability
% 2 = throttle stability
% 3 = horizontal stability

score = zeros(N,3);

% Test parameters
% Minimum altitude for considering take-off [m]
alt_take_off = 0.5;

% Limit for pitch-roll count over 5
lim_pr_count = 5;

% Limit for pitch-roll STD []
lim_pr_std = 2;

% Limit for climb rate [cm/s]
lim_climb = 200;

% Altitude thresholds
alt_1 = 1;
alt_2 = 2;

% Maximum distance for simulated data
r_max_allowed = 5;
r_final_allowed = 1;

for i = 1:N
    % Number of attempts for take off
    count_take_off = count_crossing(alt_real{i},0.5);
    % Exception: if user doesn't take off at all
    if count_take_off < 1
        fprintf('User couldnt take off for test %d - test failure\n',i)
    else
    % First test: take off, hold altitude at ~3 meters for an amount of
    % time and land, while maintaining a moderate climb rate
    % and steady pose
        % Computation of useful information
        % Maximum absolute difference
        pitch_dif = pitch_max(i) - pitch_min(i);
        roll_dif = roll_max(i) - roll_min(i);
        climb_dif = climb_max(i) - climb_min(i);
```

```
% Count of number of crossings for different (absolute) angle
% thresholds in pitch-roll
pitch_count = zeros(1,10);
roll_count = zeros(1,10);
for j = 1:10
    pitch_count(j) = count_crossing(abs(pitch_real{i}),j);
    roll_count(j) = count_crossing(abs(roll_real{i}),j);
end

if(~bool)
    % Maximum distance in meters reached from initial position
    r_max = sqrt(   max(lat_max(i),abs(lat_min(i)))^2 + ...
                    max(lon_max(i),abs(lon_min(i)))^2);

    % Final distance in meters from initial position
    r_final = sqrt(lat_scaled{i}(end)^2+lon_scaled{i}(end)^2);
end

% time spent over different altiudes
[time_over_1_i, time_over_1_f] = time_over(alt_real{i},alt_1);
[time_over_2_i, time_over_2_f] = time_over(alt_real{i},alt_2);
if ( time_over_1_i > 0 ) && ( time_over_1_f > 0 )
    time_over_1 =   (time_alt_real{i}(time_over_1_f) - ...
                    time_alt_real{i}(time_over_1_i))/1000;
else
    time_over_1 = 0;
end
if ( time_over_2_i > 0 ) && ( time_over_2_f > 0 )
    time_over_2 =   (time_alt_real{i}(time_over_2_f) - ...
                    time_alt_real{i}(time_over_2_i))/1000;
else
    time_over_2 = 0;
end

% Scoring over 10 for all values
% 1. Pitch-Roll
if pitch_count(5) <= lim_pr_count
    score_pitch = 5;
elseif pitch_count(5) >= 2*lim_pr_count
    score_pitch = 0;
else
    score_pitch = -pitch_count(5)*(5)/lim_pr_count + 10;
end

if roll_count(5) <= lim_pr_count
    score_roll = 5;
elseif roll_count(5) >= 2*lim_pr_count
    score_roll = 0;
else
    score_roll = -roll_count(5)*(5)/lim_pr_count + 10;
end

% Discount if pitch and roll have big variance (std)
if pitch_std(i) > lim_pr_std
    score_pitch = score_pitch * 0.5;
end
```

```
if roll_std(i) > lim_pr_std
    score_roll = score_roll * 0.5;
end

score(i,1) = score_pitch + score_roll;


% 2. Throttle
% Climb rate within limits
climb_mmax = max(climb_max(i),abs(climb_min(i)));
if climb_mmax > lim_climb
    score_climb = 0;
elseif climb_mmax > 0.5*lim_climb
    score_climb = 2.5;
else
    score_climb = 5;
end

% Discount for multiple attempts on take-off
if count_take_off > 1
    score_climb = max(0,score_climb - (count_take_off - 1));
end

% Maximum altitude should be between 2m and 4m (discount if more, 0 if
% <1 and >5)
if alt_max(i) > 5 || alt_max(i) < 1
    score_alt = 0;
elseif alt_max(i) > 4 || alt_max(i) < 2
    score_alt = 2.5;
else
    score_alt = 5;
end

score(i,2) = score_climb + score_alt;
if(~bool)
    % 3. Horizontal
    % r_max within boundaries (around 5 meters). decreases exponnentially
    if r_max < r_max_allowed
        score_rmax = 5;
    else
        score_rmax = 5*exp(-0.5*(r_max - r_max_allowed));
    end

    % r_final below 1m from origin. decreases exponnentially
    if r_final < r_final_allowed
        score_rfinal = 5;
    else
        score_rfinal = 5*exp(-(r_final - r_final_allowed)^2);
    end

    score(i,3) = score_rmax + score_rfinal;
end
% Time spent over a minimum of 1m should be at least 15 seconds
% and over 2m at least 10 seconds, and affects overall score
if time_over_1 < 15
    score(i,:) = 0.5*score(i,:);
elseif time_over_2 < 10
```

```
            score(i,:) = 0.75*score(i,:);
        end

    end
end

%% Parameter calculation based on scores obtained

% Removal of scores for failed tests.
% If all tests are scored 0, we leave it as it is
i = 1;
M = N;
while( i <= M )
    if max(score(i,:)) == 0
        if size(score,1) > 1
            score = [score((1:i-1),:);score((i+1:end),:)];
            M = M - 1;
            i = 1 ;
        else
            break
        end
    else
        i = i + 1;
    end
end

params = zeros(1,2);
w_rp = 0.6;
w_th = 0.4;
w_gps = 1 - w_rp - w_th;

weights = [w_rp, w_th, w_gps]; %weight of each score

score_wmean = dot(weights,mean(score,1));

% If overall score is over 8, no need for assistant
if score_wmean >= 8
    params(1) = 0;
    params(2) = 0;
elseif score_wmean >=6
    % If 6<=score<8, flat values for assistant parameters
    params(1) = 1000;   % 10 saturation
    params(2) = 10;     % 10/sec rate saturation
elseif score_wmean >= 2
    % If 2<=score<6, linear function decreasing from max to min values
    params(1) = (score_wmean - 2)*(700/4) + 300;
    params(2) = (score_wmean - 2)*(5/4)+5;
elseif score_wmean > 0
    % If score < 2, min values for assistant parameters
    params(1) = 300;    % 3 saturation
    params(2) = 5;      % 5/sec rate saturation
else
    % score = 0 means user cant fly a drone and has to practice in a
    % simulator
    params(1) = -1;
    params(2) = -1;
end
```

```
%% Score Display
score_mean = mean(score,1);

fprintf('El puntaje ponderado total obtenido para la prueba es de:\n')
fprintf('Estabilidad angular:\t\t%2.1f / 10\n',score_mean(1))
fprintf('Aceleraci□t\t\t\t%2.1f / 10\n',score_mean(2))
fprintf('Posicionamiento:\t\t\t%2.1f / 10\n',score_mean(3))
fprintf('Puntaje Final: %2.1f\n\n',score_wmean)

score_mean2 = weights*(score');
[score_max,imax] = max(score_mean2);
[score_min,imin] = min(score_mean2);

fprintf('Mejor prueba:\tPrueba %d; \tPuntaje = %2.1f\n',imax,score_max)
fprintf('Peor prueba:\tPrueba %d; \tPuntaje = %2.1f\n\n',imin,score_min)
if params > 0
    fprintf('Con estos puntajes, calibrar los par□tros con los siguientes
valores:\n')
    fprintf('sat_angles = %g\n',params(1))
    fprintf('sat_angle_deriv = %g\n',params(2))
elseif params < 0
    disp('Con estos puntajes, se recomienda comenzar por practicar en un simulador')
else
    disp('Con estos puntajes, no necesita un asistente de vuelo')
end
```

Anexo VI - Función **length_latdeg.m**

```
function l = length_latdeg(lat_deg)
% function length_latdeg
%   calculates the approximate length, in meteres, of 1 of latitude
%
%   use:
%       l = length_latdeg(lat_deg)
%   inputs:
%       lat_deg:    latitude, in decimal degrees, where we want to calcule
%                   the length of 1
%   outputs:
%       l:          approximate length, in meters.
%
%   this function calculates the approximate length of 1 of latitude
%   according to the WGS84 model of the earth, where it is modeled by
%   an elipsoide, with semi-axes:
%   a = 6,378,137 [m],      semi-major axe
%   b = 6,356,752.3142 [m]  semi-minor axe
%
%   the formula used for 1 of latitude:
%
%       l = pi*a*(1-e^2)/(180*(1-e^2*(sin(lat_rad))^2)^(3/2))
%
%   where
%       lat_rad             is the current latitude in radians
%       e^2 = (1-b^2/a^2)   is the earth's excentricity
%
%   reference:
%

a = 6378137;        % 6,378,137 meters
b = 6356752.3142;    % 6,356,752.3142 meters

e2 = (a^2-b^2)/a^2;
fact = pi*a*(1-e2)/180;

lat_rad = lat_deg*pi/180;

l = fact / (1-e2*(sin(lat_rad))^2)^(3/2);


end
```

Anexo VII – Función **length_londeg.m**

```
function l = length_londeg(lat_deg)
% function length_londeg
%   calculates the approximate length, in meteres, of 1 of longitude
%
%   use:
%       l = length_londeg(lat_deg)
%   inputs:
%       lat_deg:    latitude, in decimal degrees, where we want to calcule
%                   the length of 1
%   outputs:
%       l:          approximate length, in meters.
%
%   this function calculates the approximate length of 1 of longitude
%   according to the WGS84 model of the earth, where it is modeled by
%   an elipsoide, with semi-axes:
%   a = 6,378,137 [m],      semi-major axe
%   b = 6,356,752.3142 [m]  semi-minor axe
%
%   the formula used for 1 of longitude:
%
%       l = pi*a*cos(lat_rad)/(180*(1-e^2*(sin(lat_rad))^2)^(1/2))
%
%   where
%       lat_rad             is the current latitude in radians
%       e^2 = (1-b^2/a^2)   is the earth's excentricity
%
%   reference:
%


% WGS84
a = 6378137;        % 6,378,137 meters
b = 6356752.3142;   % 6,356,752.3142 meters

e2 = (a^2-b^2)/a^2;
fact = pi*a/180;

lat_rad = lat_deg*pi/180;

l = fact * cos(lat_rad) / (1-e2*(sin(lat_rad))^2)^(1/2);

end
```

Anexo VIII - Función **count_crossing.m**

```
function n = count_crossing(signal,threshold)
% function count_crossing
%   counts how many times a signal crosses a threshold further away from
%   zero
%
%   use:
%       n = count_crossing(signal,threshold)
%   inputs:
%       signal:     1-D vector containing a signal
%       threshold:  limit value where we count crossing of the signal
%   outputs:
%       n:          number of times the signal crosses the threshold
%
%   this function counts the number of times the signal crosses the
%   threshold, further away from zero. if threshold is a positive number,
%   the formula used is:
%       if signal(t+1) > threshold and signal(t) < threshold ++n
%   otherwise, if the threshold is a negative number, the formula used is:
%       if signal(t+1) < threshold and signal(t) > threshold ++n
%


n = 0;
if threshold > 0
    for i= 1:length(signal)-1
        if signal(i+1) >= threshold
            if signal (i) < threshold
                n = n + 1;
            end
        end
    end
else
    for i= 1:length(signal)-1
        if signal(i+1) <= threshold
            if signal (i) > threshold
                n = n + 1;
            end
        end
    end
end
```

Anexo IX – Función **time_over.m**

```
function [t1,t2] = time_over(signal,threshold)
% function time_over:
%   checks how long a signal tresspasses a given threshold
%
%   use:
%       [t1,t2] = time_over(signal,threshold)
%   inputs:
%       signal:     1-D vector containing a signal
%       threshold:  limit value
%   outputs:
%       t1,t2:          index where the signal tresspassed the threshold
%
%   this function checks the index of the signal where it is greater than
%   a given threshold. it always returns the last interval where the
%   signal is greater than the argument. it only works for positive
%   signals.
%

t1 = 0;
t2 = 0;
found = 0;

l = length(signal);

if l < 2
    return;
end

for t = 2:l
    if signal(t) >= threshold && signal(t-1) < threshold
        t1 = t;
        break
    end
end

if t1 == 0
    return;
else
    for t = t1+1:l
        if signal(t) < threshold
            t2 = t;
            found = 1;
            break;
        end
    end
end

if found
    signal_2 = signal(t2:end);
    [t11,t22] = time_over(signal_2,threshold);
    if t11 > 0
        if t22-t11 > t2-t1
            t1 = t2+t11-1;
            t2 = t2+t22-1;
        end
```

```
    end
end

end
```

Anexo X – Programa **split_logs.m**

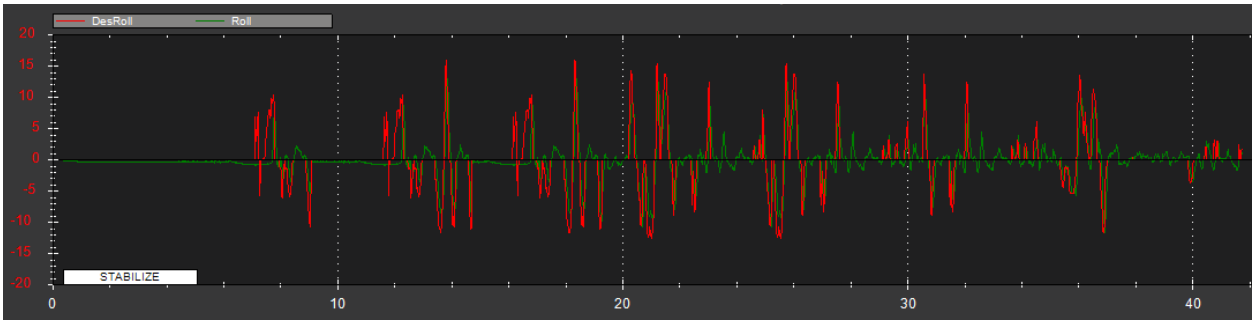
```
function split_logs(user)
% function to split vectors from dataflash logs
% will work on attitude and altitude vectors

i = 1;
while(true)
    try
        load(fullfile(user,strcat('data_dflog',num2str(i),'.mat')));
    catch err
        if (i == 1)
            error(strcat('no data found for ',user,' user'));
        else
            break;
        end
    end

    time_att = ATT(:,2);
    roll = ATT(:,4);
    pitch = ATT(:,6);
    yaw = ATT(:,8);

    time_alt = CTUN(:,2);
    alt = CTUN(:,7);
    climb = CTUN(:,12);

    roll_out{i} = split_signal(roll,time_att);
    pitch_out{i} = split_signal(pitch,time_att);
    yaw_out{i} = split_signal(yaw,time_att);
    time_att_out{i} = split_signal(time_att,time_att);

    alt_out{i} = split_signal(alt,time_alt);
    climb_out{i} = split_signal(climb,time_alt);
    time_alt_out{i} = split_signal(time_alt,time_alt);

    N_out(i) = length(roll_out{i});
    i = i+1;
end
L = i-1;

% store data for analysis

mkdir(strcat(user,'\real'));

for i = 1:L
    for j = 1:N_out(i)
        roll_log = roll_out{i}{j};
        pitch_log = pitch_out{i}{j};
        yaw_log = yaw_out{i}{j};
        time_att_log = time_att_out{i}{j};

        alt_log = alt_out{i}{j};
        climb_log = climb_out{i}{j};
        time_alt_log = time_alt_out{i}{j};
        k = j + sum(N_out(1:(i-1)));
        save(fullfile(user,'real',strcat('log_real',num2str(k),'.mat')),...
```

```
            'roll_log','pitch_log','yaw_log','time_att_log','alt_log',...
            'climb_log','time_alt_log');
    end
end
```

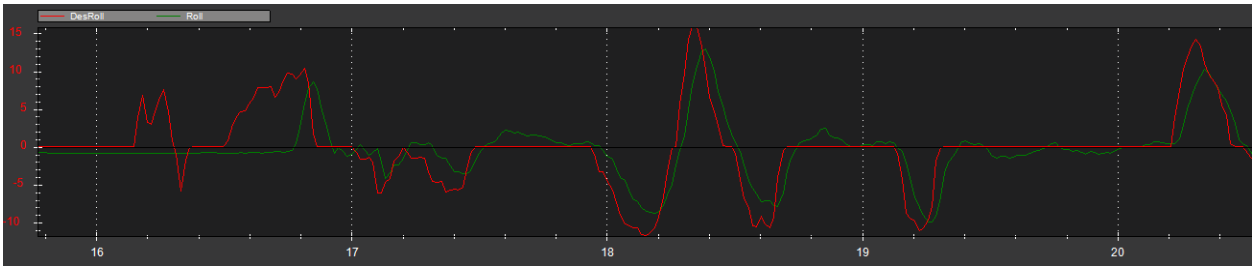
Anexo XI – Gráficos de los ángulos de alabeo (*roll*), elevación (*pitch*) y dirección (*yaw*) en las pruebas básicas de desempeño, en conjunto con las referencias pedidas por el operario. En cada caso se muestra la señal completa y un acercamiento a una zona específica para ilustrar la calidad de respuesta de la nave.

Roll:

Completo:

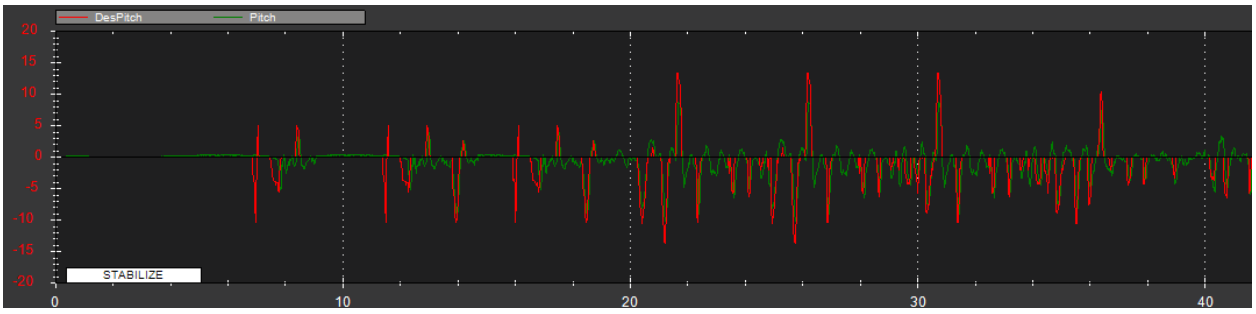


Acercamiento:

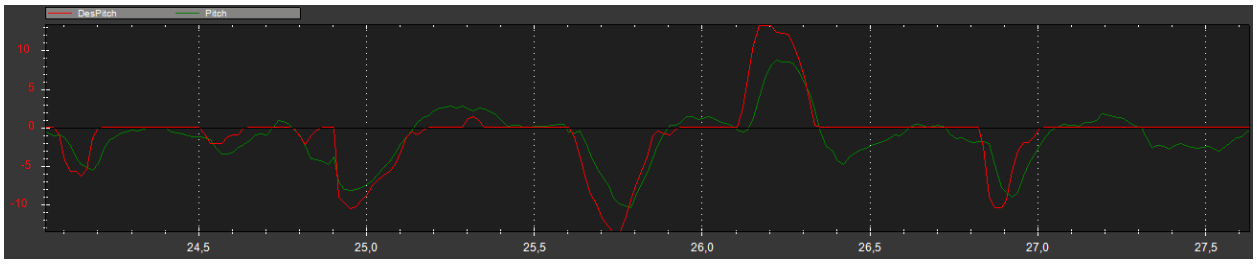


Pitch:

Completo:

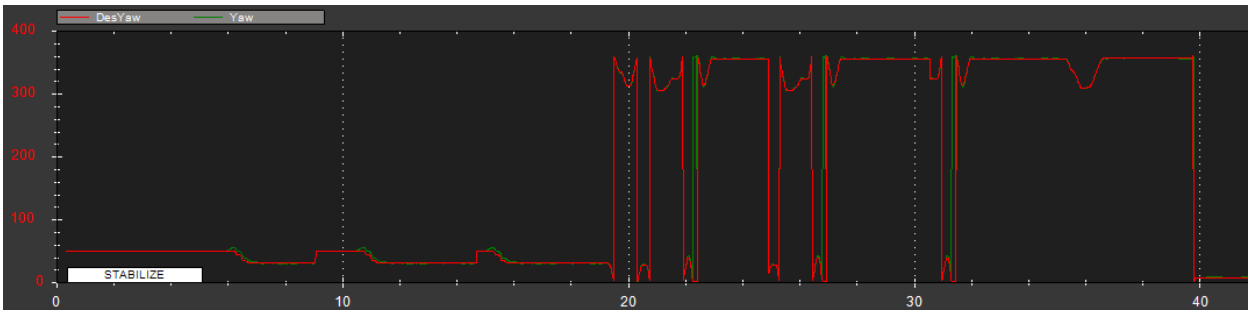


Acercamiento:

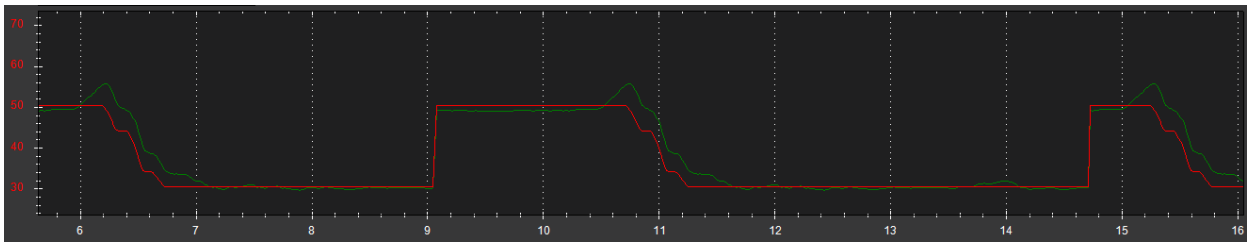


Yaw:

Completo:

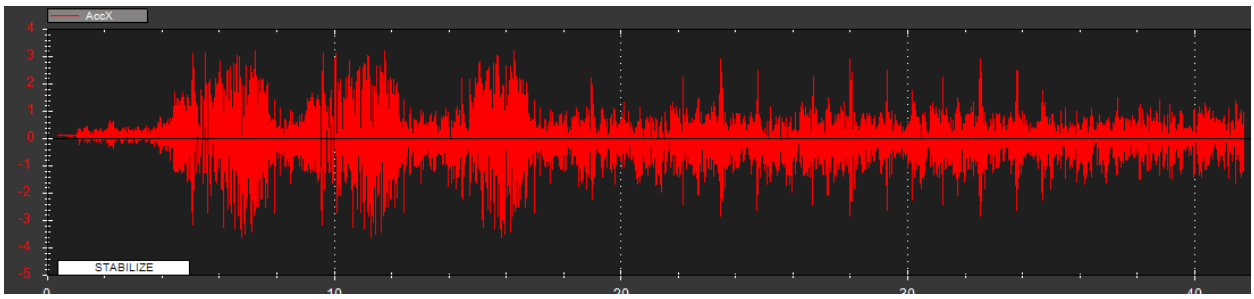


Acercamiento:

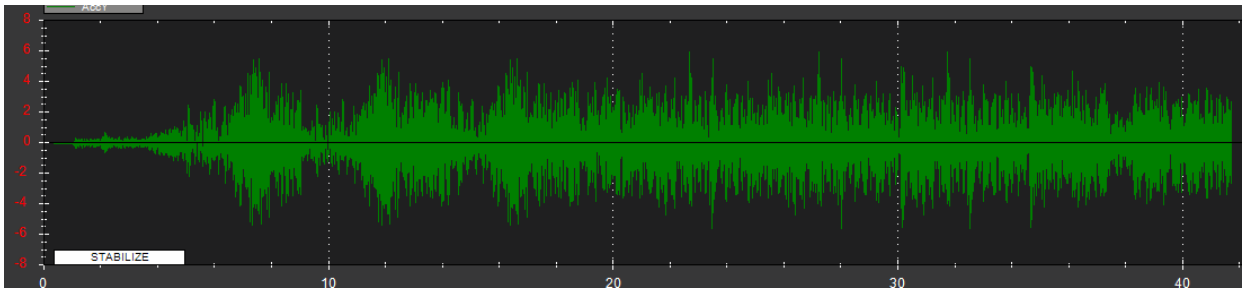


Anexo XII – Gráficos de las mediciones obtenidas por los acelerómetros en los 3 ejes en la prueba básica de desempeño.

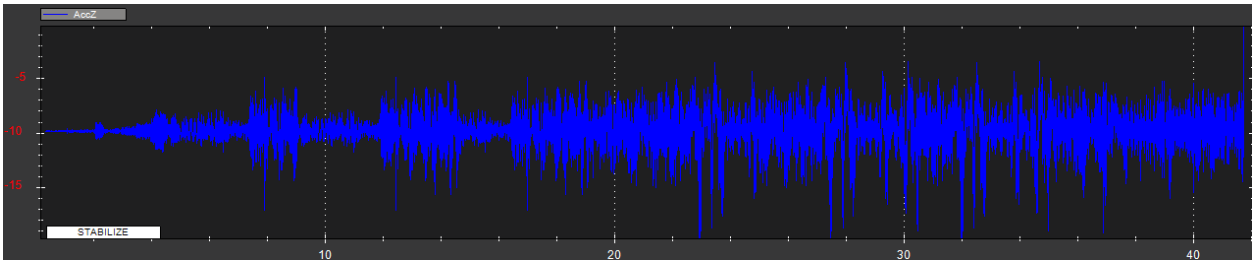
Eje X:



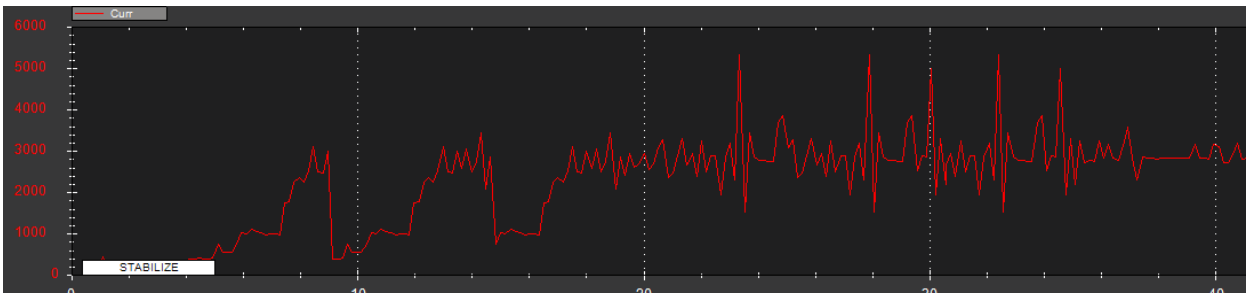
Eje Y:



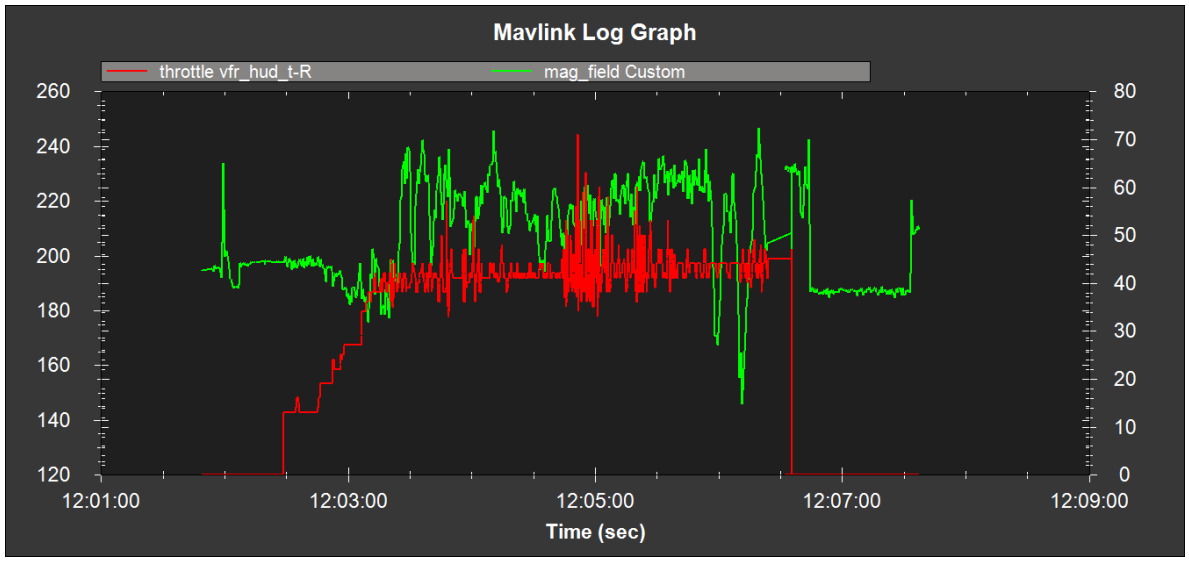
Eje Z:



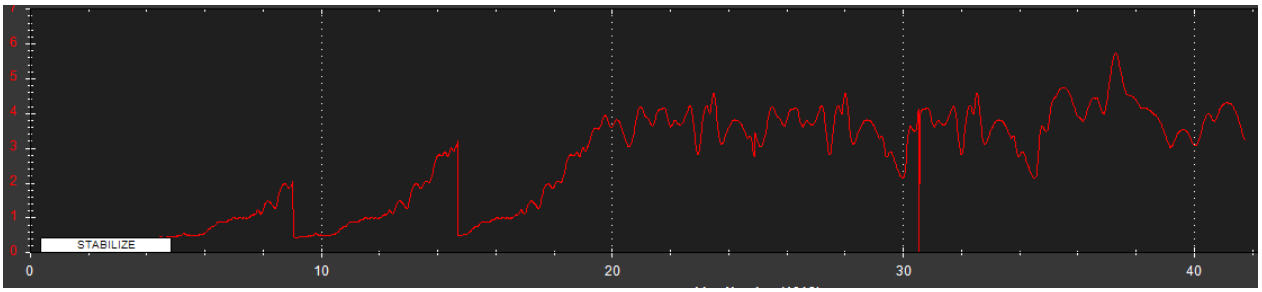
Anexo XIII – Gráfico de la corriente circulante, en unidades de $[A] \cdot 100$.



Anexo XIV – Gráfico completo de campo magnético total detectado por el compás (verde, eje izquierdo) en conjunto con la aceleración pedida en porcentaje (rojo, eje derecho).



Anexo XV – Gráfico completo de la altitud medida, en metros.

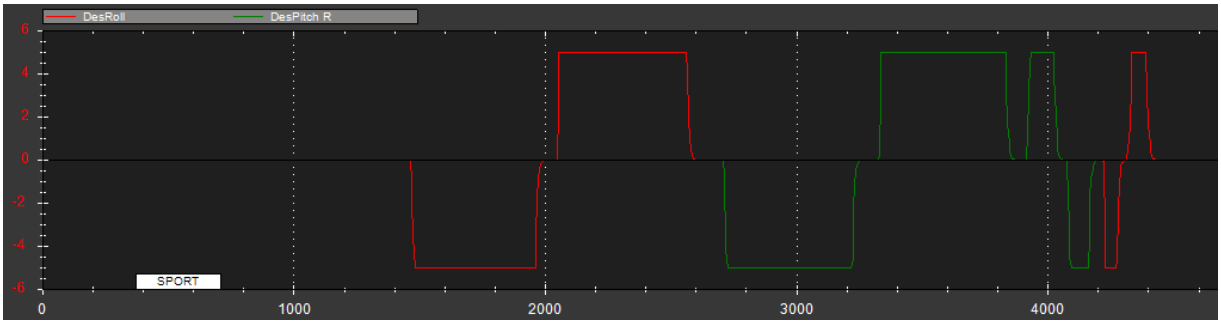


Anexo XVI – Gráficos de las referencias calculadas a partir de las entradas PWM de los ángulos de elevación y alabeo (*pitch-roll*).

Primera prueba: saturación lineal.

Valores de parámetros:

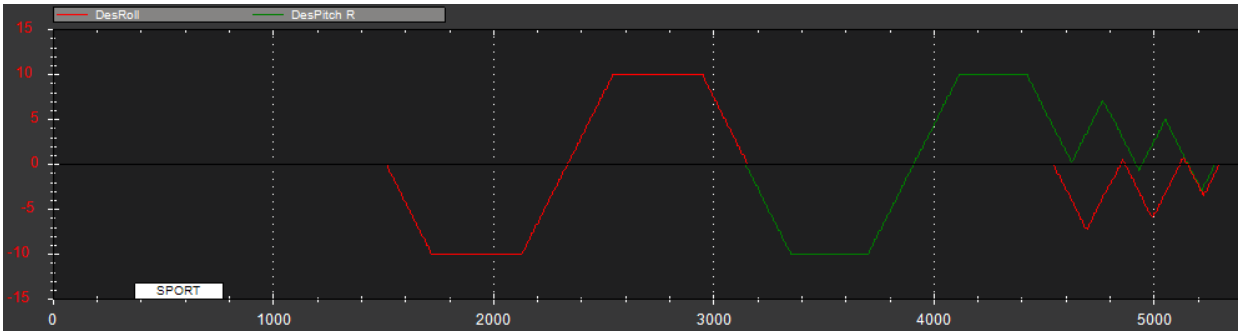
- `sat_angles = 500`
- `sat_angle_deriv = 1000`



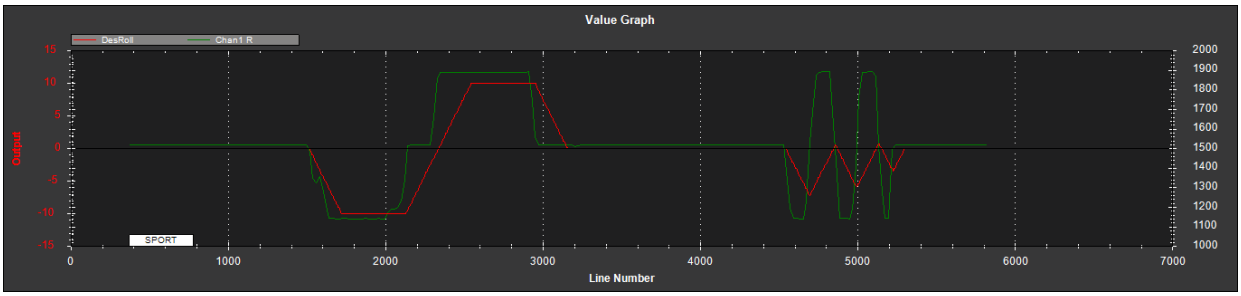
Segunda prueba: saturación lineal y de tasa de cambio.

Valores de parámetros

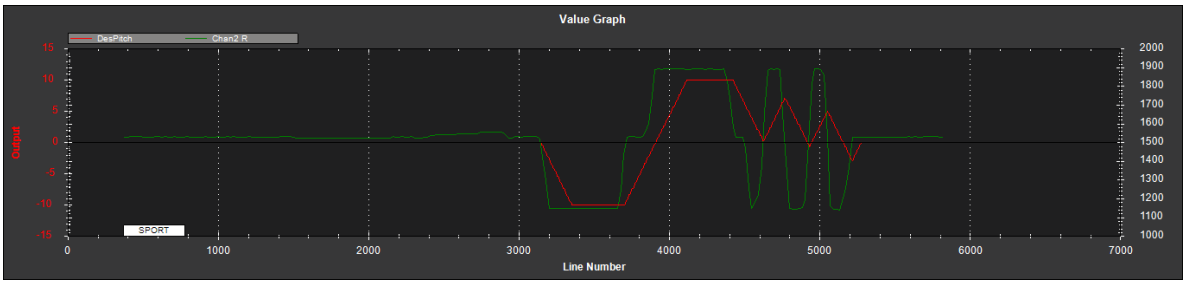
- `sat_angles = 1000`
- `sat_angle_deriv = 10` $\Longrightarrow 10° / s$ `máximo`



Ángulo de alabeo (*roll*) en grados y canal 1 del mando RC en [ms] (PWM).

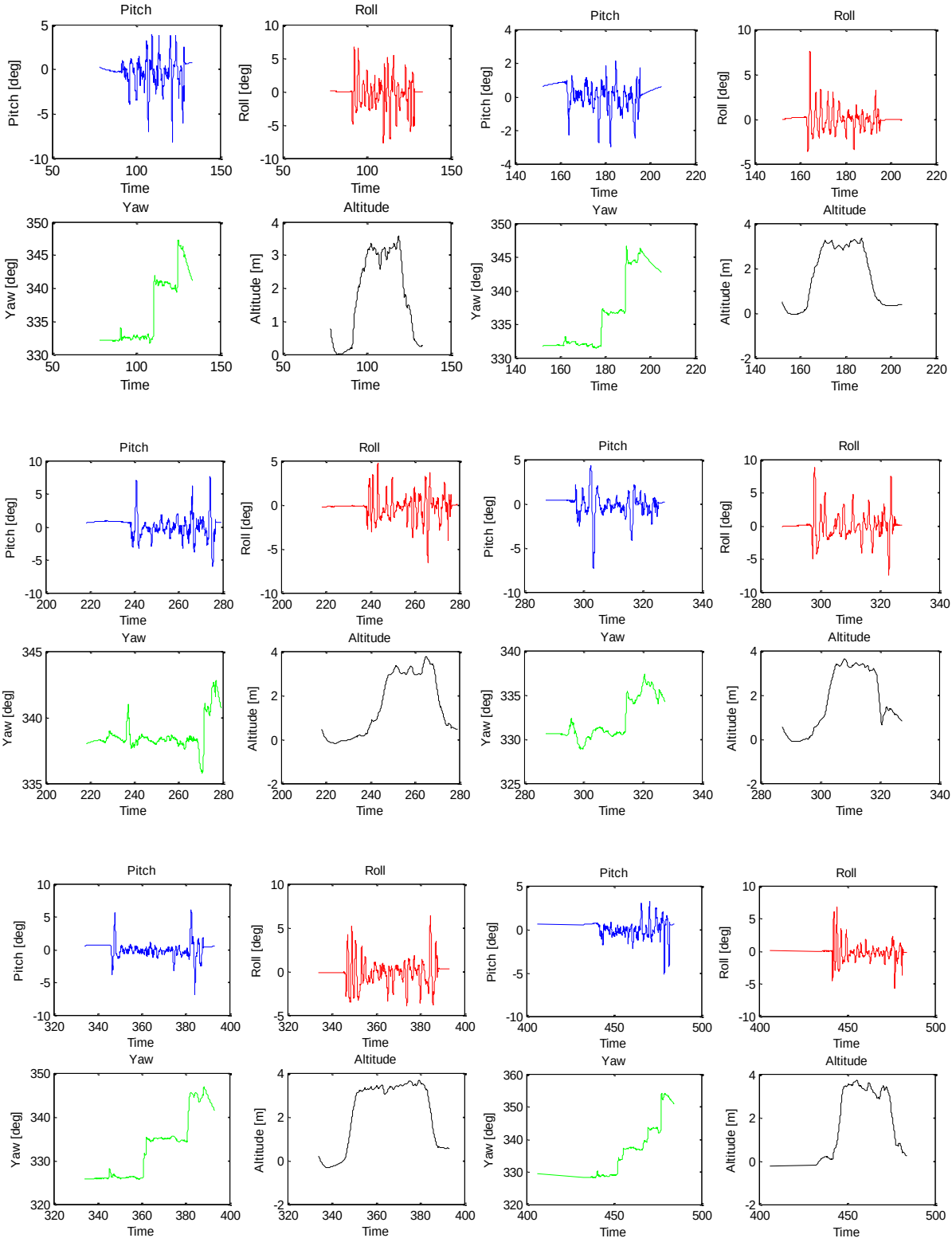


Ángulo de elevación (*pitch*) en grados y canal 2 del mando RC en [ms] (PWM).

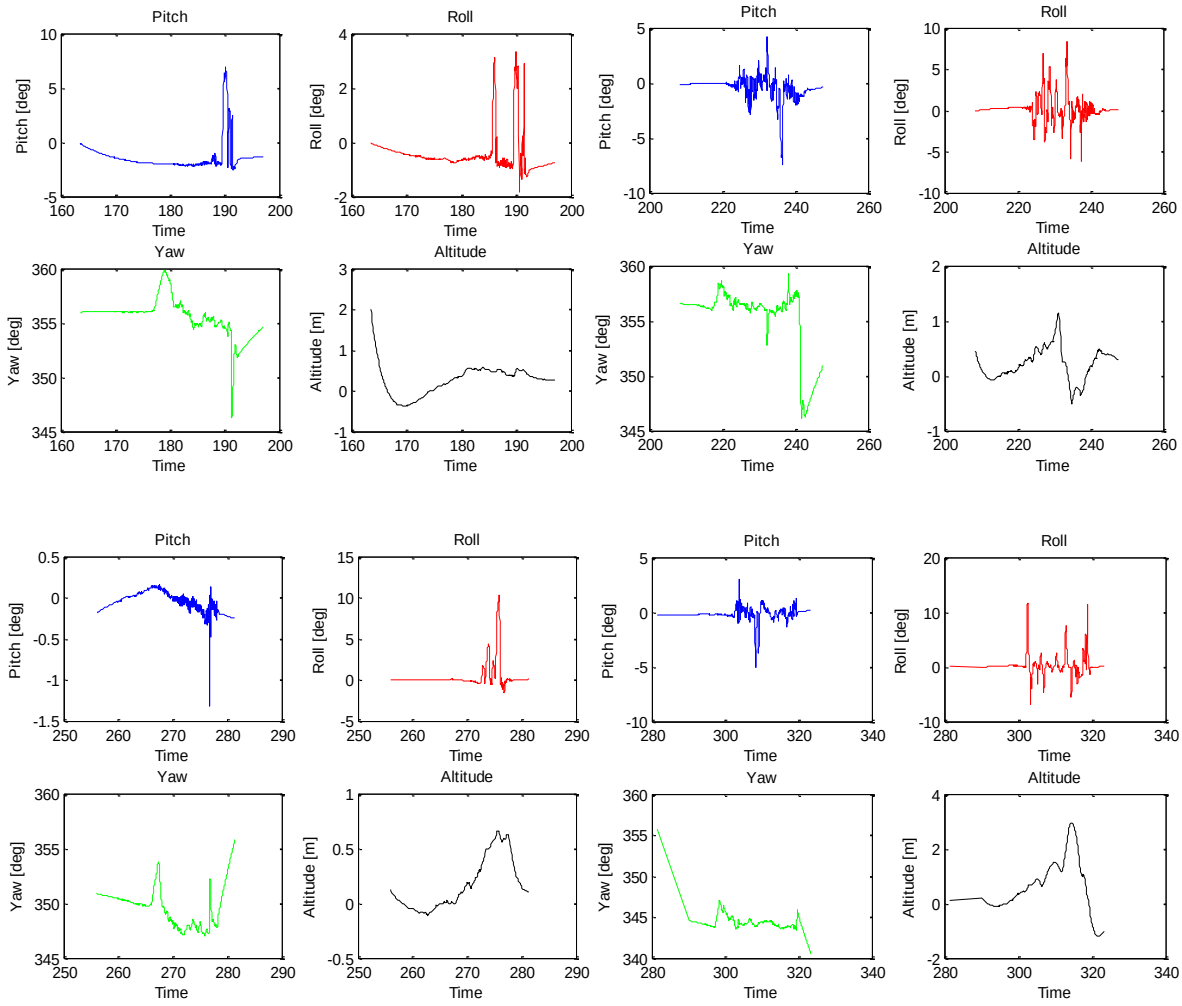


Anexo XVII – Gráficos de las pruebas de operación para cada operario.

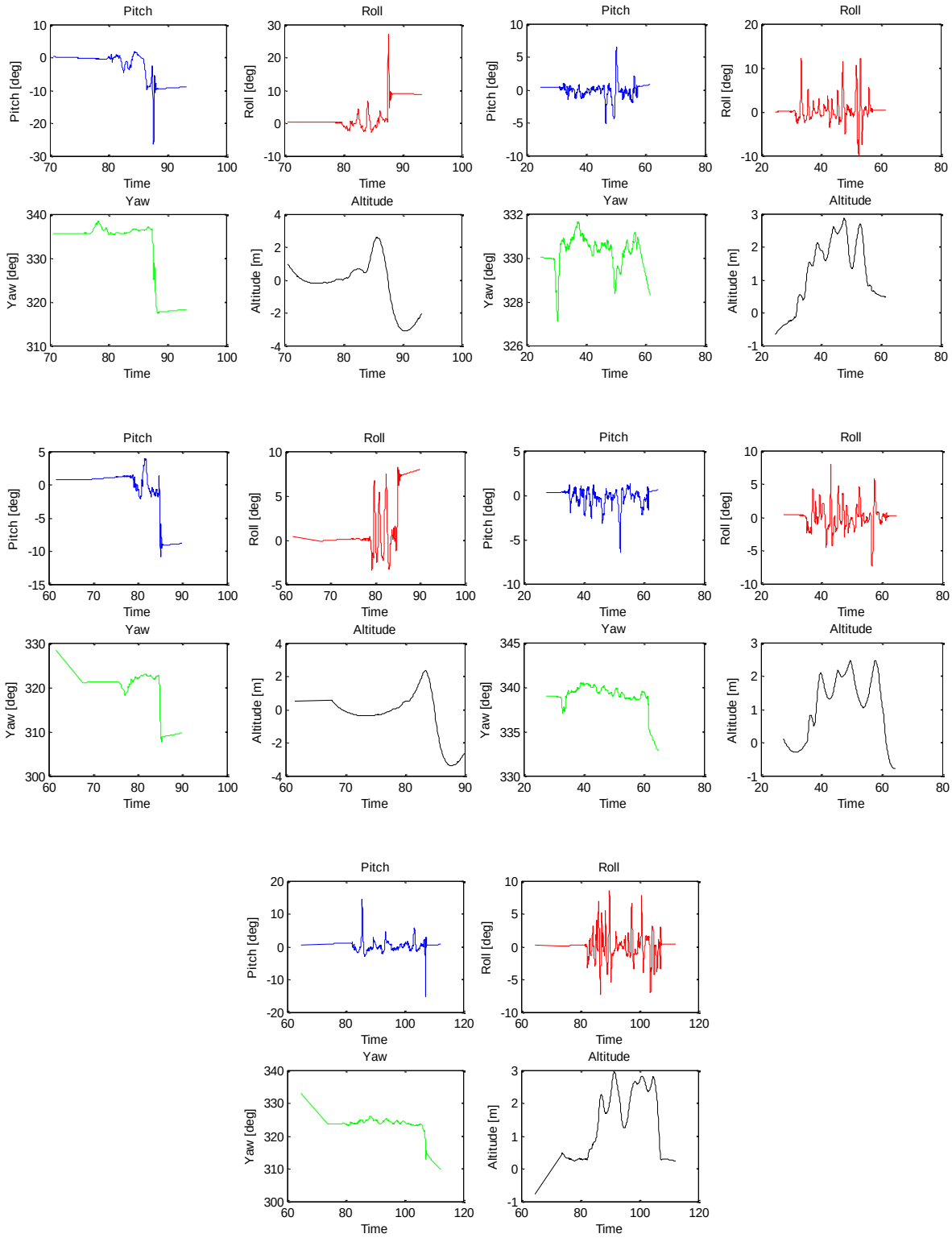
Primer operario:



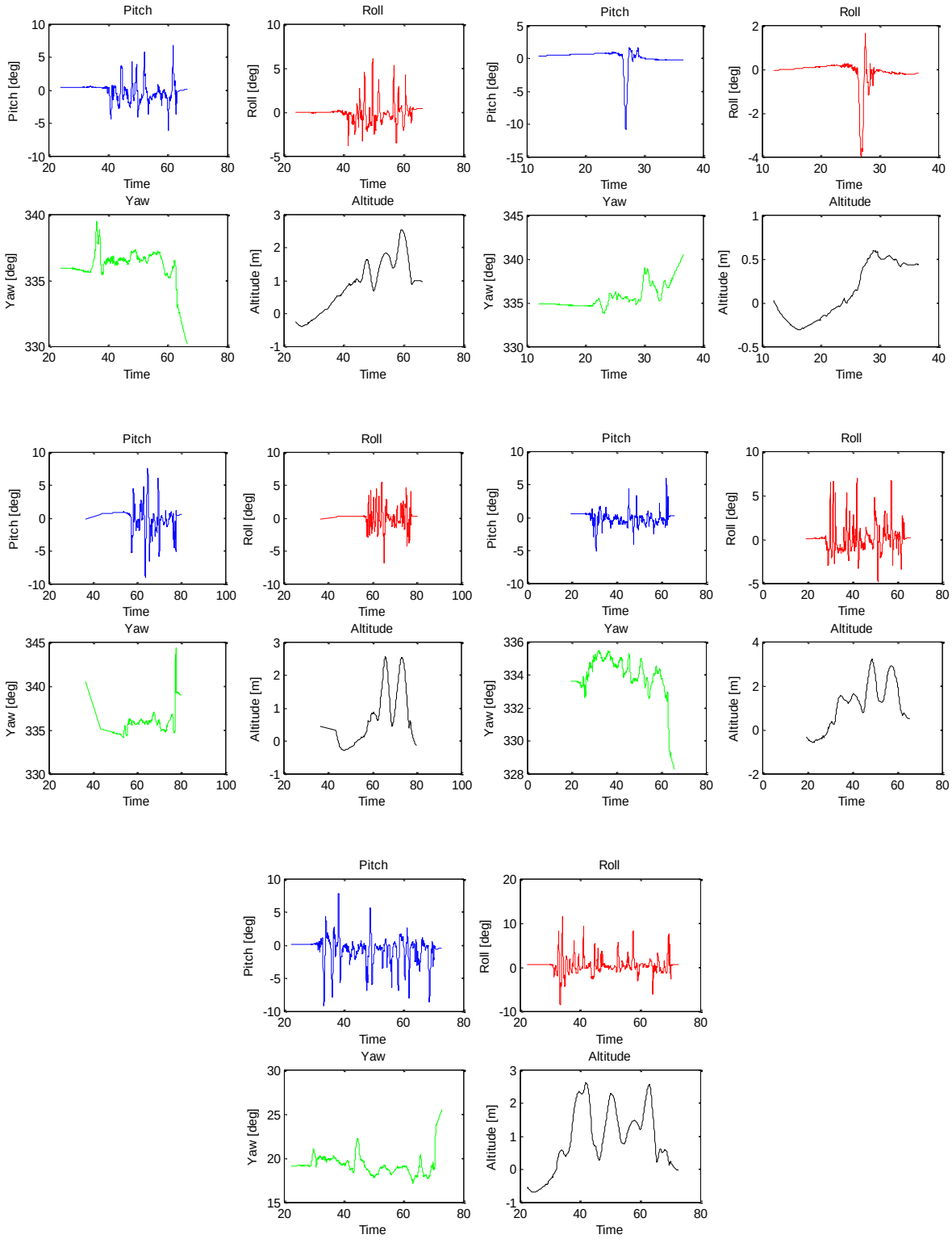
Segundo Operario:



Tercer Operario:

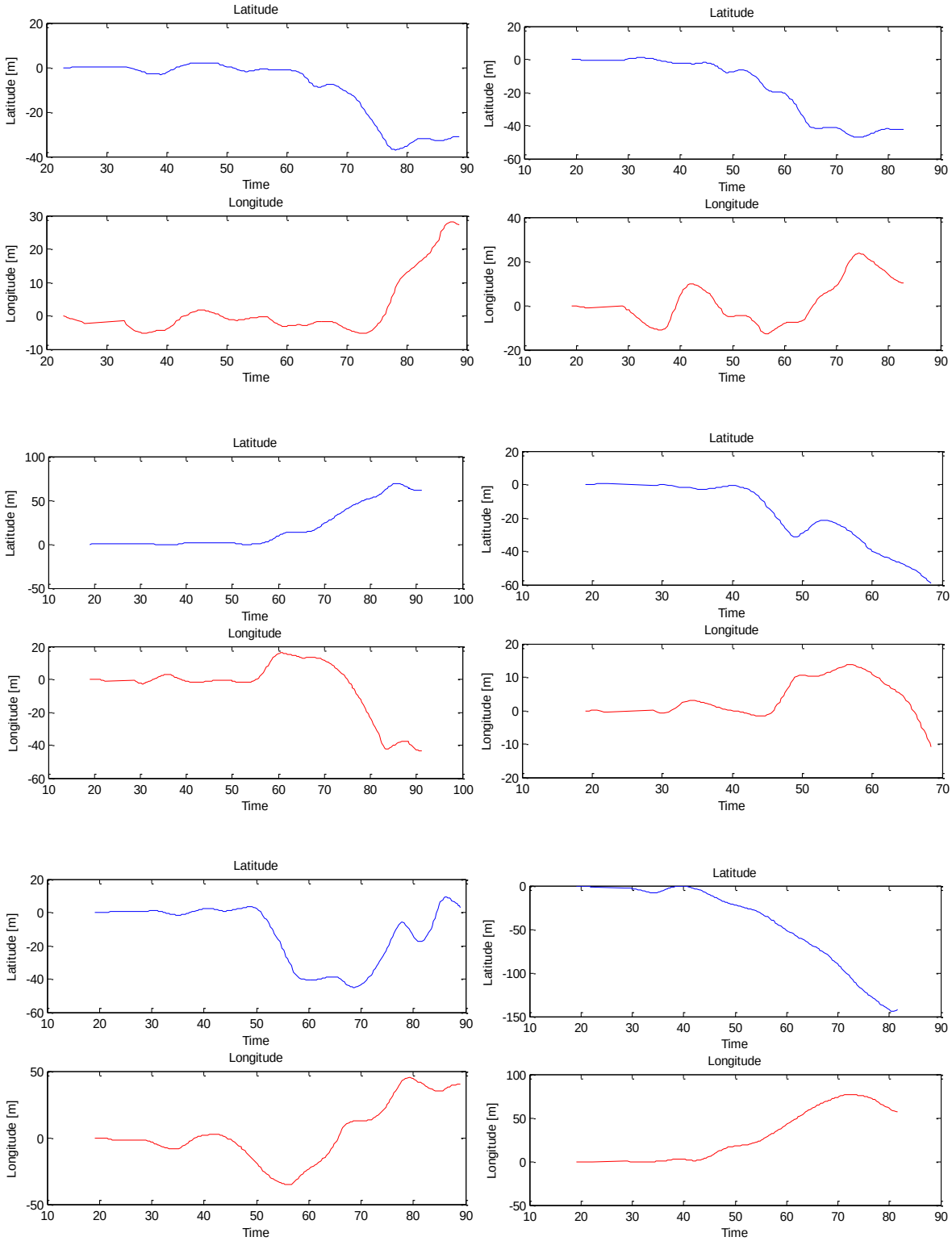


Cuarto Operario:

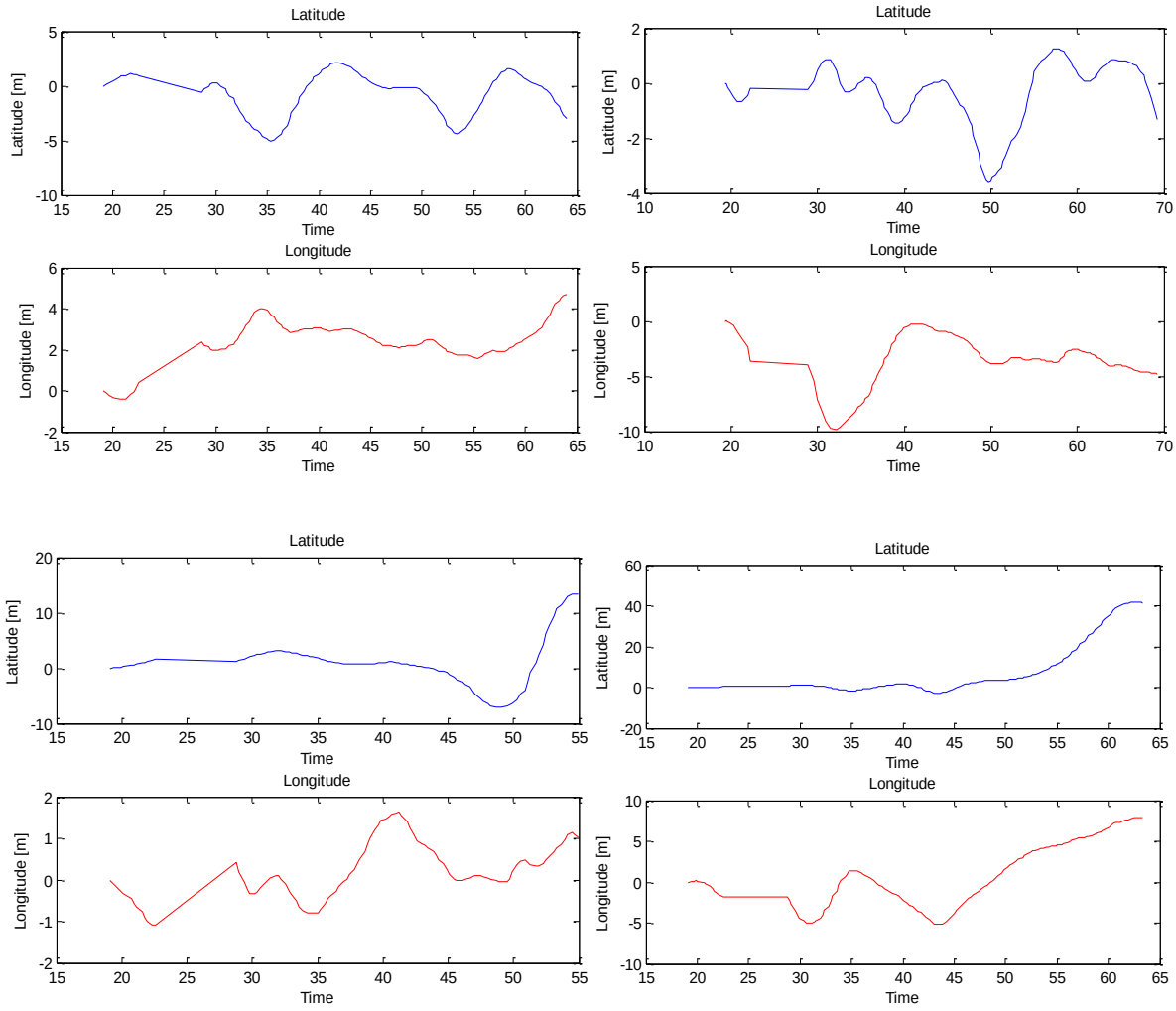


Anexo XVIII – Resultados de las simulaciones para las pruebas de cada operario.

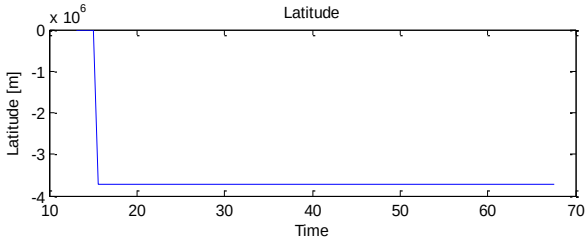
Primer operario:

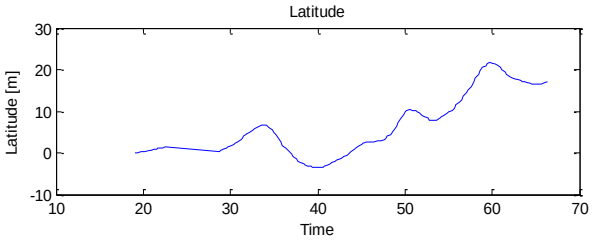


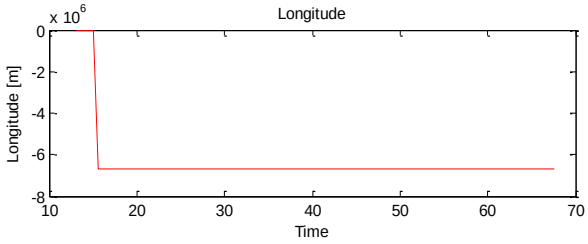
Segundo operario:

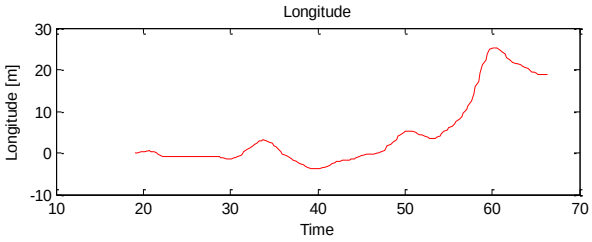


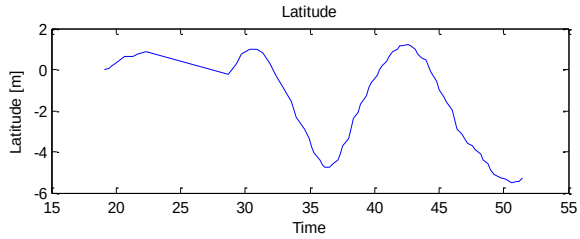
Tercer operario:

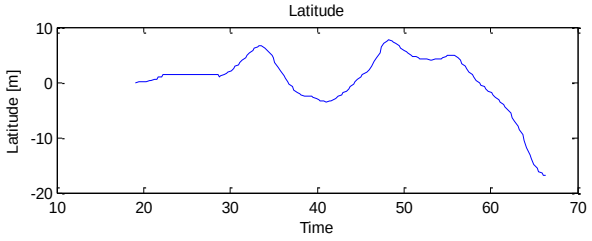


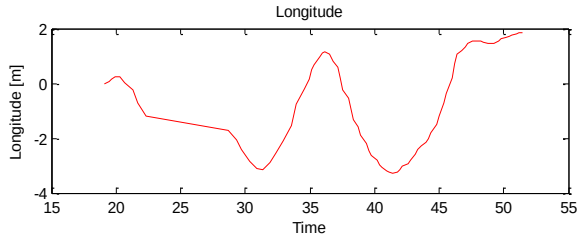


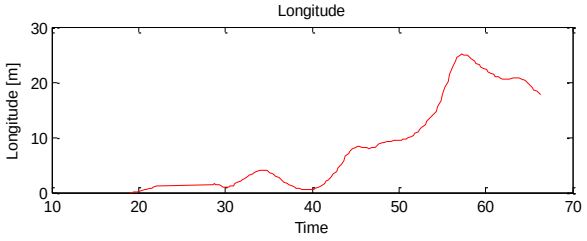


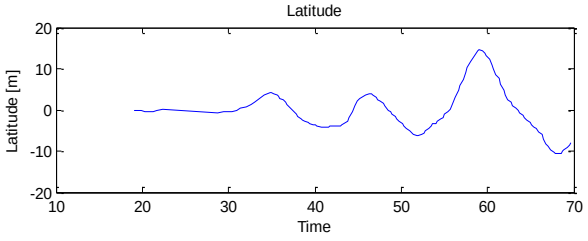


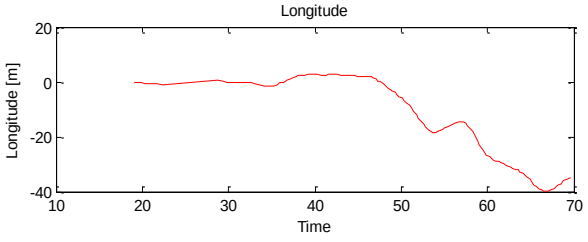




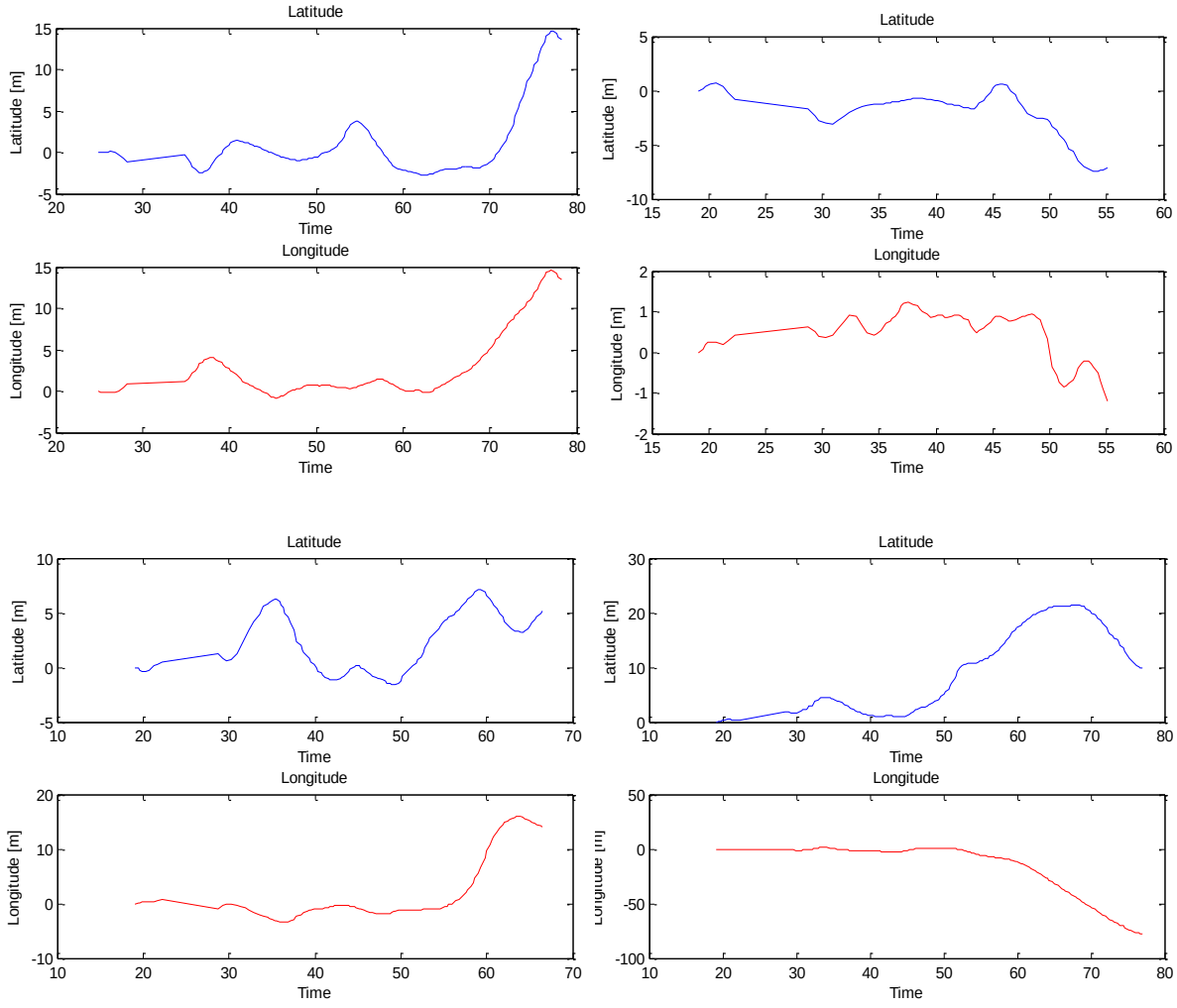


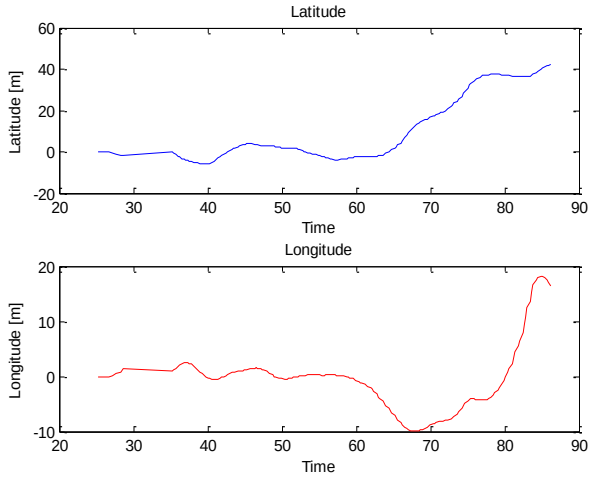






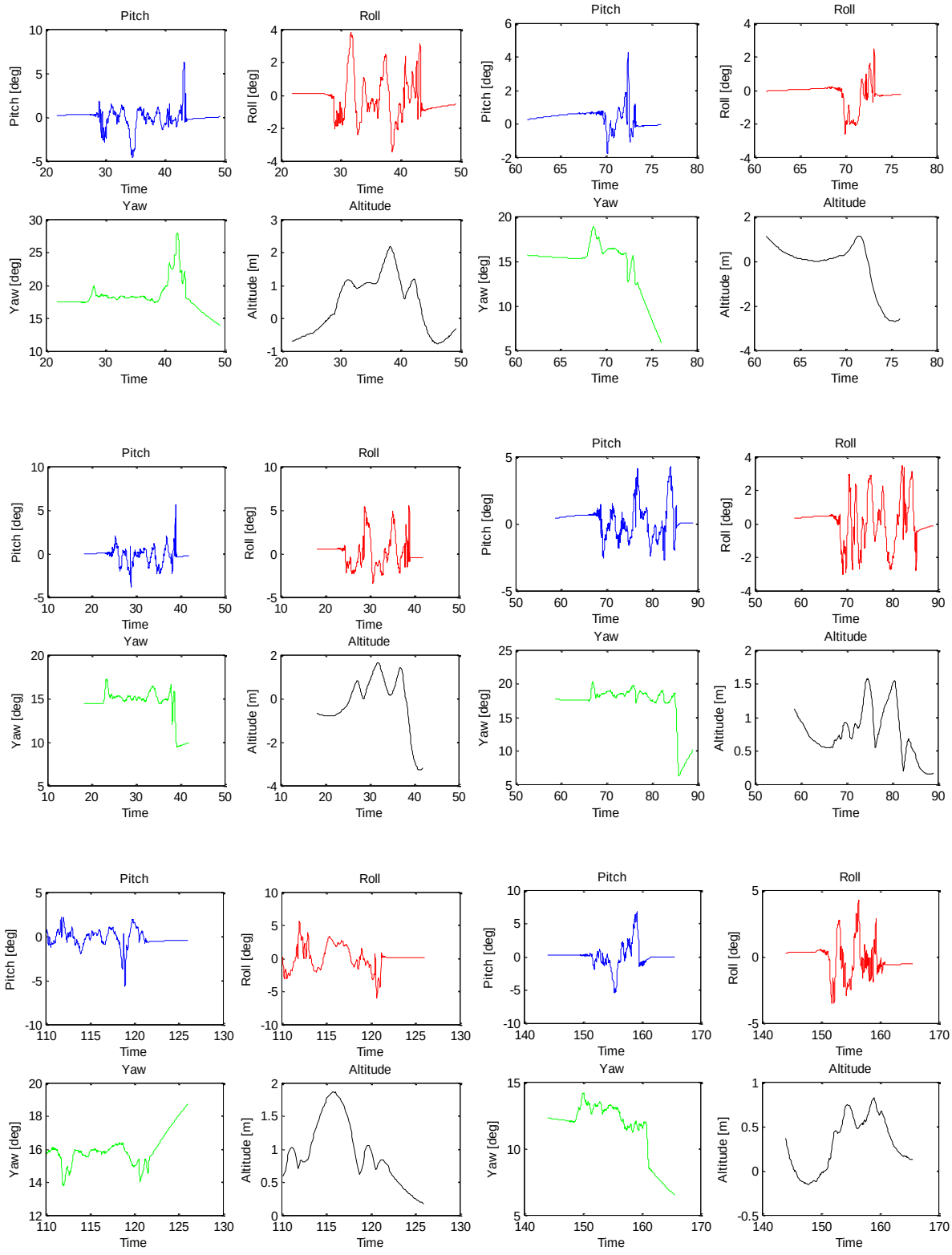
Cuarto operario:

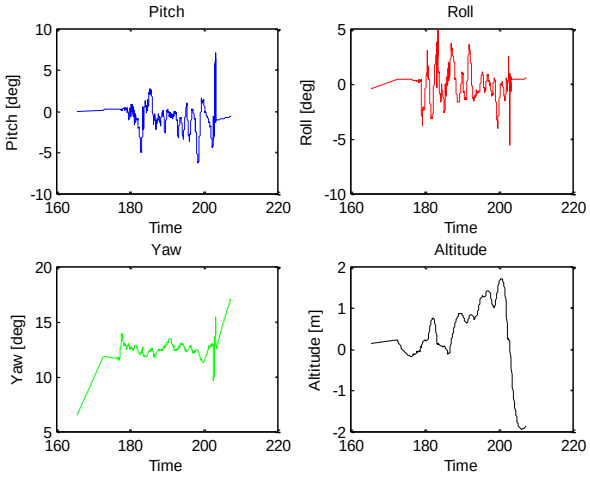




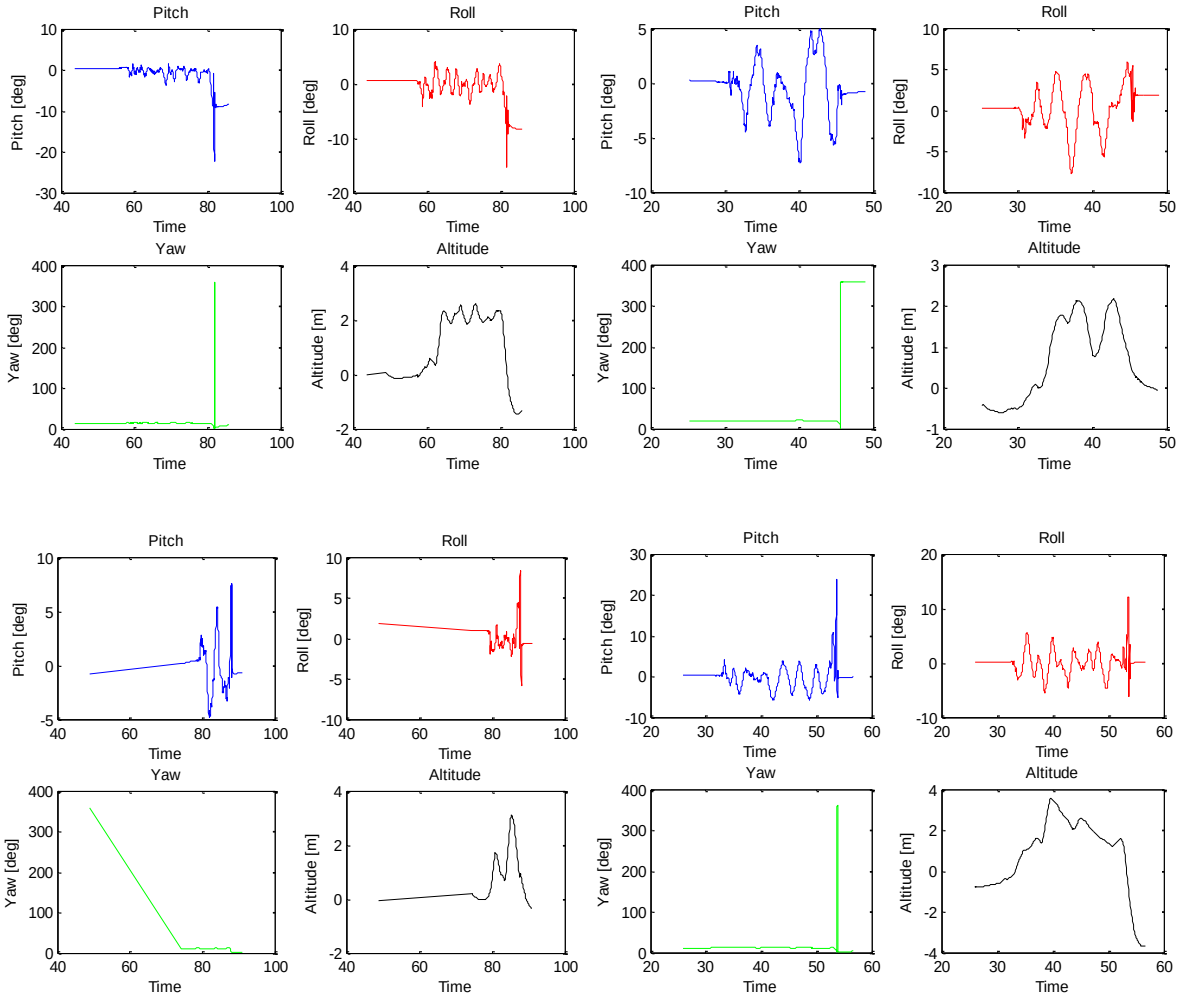
Anexo XIX – Pruebas con asistente activado.

Segundo operario:




Pitch
Pitch [deg]
10
5
0
-5
-10
160
180
200
220
Time
Roll
Roll [deg]
5
0
-5
-10
160
180
200
220
Time
Yaw
Yaw [deg]
20
15
10
5
160
180
200
220
Time
Altitude
Altitude [m]
2
1
0
-1
-2
160
180
200
220
Time

Tercer operario:



Cuarto operario: